Introduzione al VBA

Microsoft Excel è un foglio di calcolo che fornisce strumenti semplici e avanzati per creare e gestire qualsiasi tipo di elenco, inoltre per offrire uno strumento professionale ***Microsoft*** ha introdotto nel pacchetto Office, e su Excel in particolare, la possibilità di sfruttare la potenza di un vero e proprio linguaggio di programmazione ad interazione con il foglio di calcolo. Da ciò nasce il ***Visual Basic for Application***, noto semplicemente come ***VBA*** che è una evoluzione del linguaggio **Basic,** in dotazione ai personal computer nei primi anni 80, e consente di scrivere codice in grado di eseguire automaticamente azioni su un documento e/o sul suo contenuto.

Lo scopo della programmazione in **VBA** è di usare i dati presenti nel foglio di calcolo come input per facilitare le operazioni necessarie per ottenere un risultato, sia esso di calcolo che di ricerca, diventando così potenti strumenti di lavoro. In pratica, attraverso un semplice ***Click*** su un pulsante, il computer elabora una serie di dati, agevolando così il nostro lavoro, utilizzando un sotto programma sviluppato tramite VBA che, acquisendo i dati presenti nel foglio di calcolo, restituirà sempre sullo stesso foglio il risultato derivante dai calcoli che avrete scelto di far eseguire.

Sulla disponibilità e l'uso di VBA si devono fare alcune puntualizzazioni:

1. VBA, come tutti i linguaggi di programmazione, ha una propria sintassi e non tollera errori, uno spazio omesso o fuori luogo, un punto invece di una virgola e verrà rimandato un errore.
2. Il codice che viene prodotto è immagazzinato nel file dell'applicazione che è stata utilizzata per lo sviluppo e non in un file separato, questo spiega anche il fatto che ***non è possibile*** effettuare un programma eseguibile indipendente con VBA, in quanto il codice sviluppato sarà eseguito attraverso il programma host utilizzato (Excel, Word etc.) per la sua scrittura.
3. Da un codice in Excel è possibile gestire documenti Word o presentazioni PowerPoint

Si deve tener presente che VBA è organizzato in un modello gerarchico di elementi (tipo le matriosche) ed è un contenitore per oggetti che possiamo manipolare tramite le loro proprietà (es. il colore), i metodi (es. aggiunta di un elemento) ed eventi (es. risposta a un clic del mouse). Pertanto possiamo considerare Il Visual Basic for Application lo strumento ideale per la creazione di veri e propri applicativi di Office Automation.

Fondamenti di Microsoft Excel

Abbiamo visto che l'ambiente di programmazione Visual Basic, dipende da Excel, di conseguenza, per utilizzarlo è necessario prima aprire Excel cliccando su **Start - (Tutti i) Programmi - Microsoft Office - Microsoft Office Excel**, oppure se si dispone di un documento di Excel in Esplora risorse di Windows, nella cartella Documenti, sul desktop, etc. è possibile, facendo doppio clic su di esso, avviare Excel e aprire il documento stesso. A tal proposito si deve considerare che i file che contengono codice VBA, dalla versione successiva a Office 2007, vengono distinti dagli altri con un'estensione e un'icona diversa

Fig. 1

A sinistra è rappresentata l'icona di un file che **NON** contiene codice VBA e ha estensione **.xlsx**, mentre a destra è raffigurata l'icona di un file che contiene codice VBA con estensione **.xlsm**. Inoltre anche il formato di salvataggio è diverso, infatti quando viene salvato un file che contiene codice VBA occorre indicare a Excel che deve salvarlo usando il formato *Cartella di lavoro con attivazione di macro di Excel*

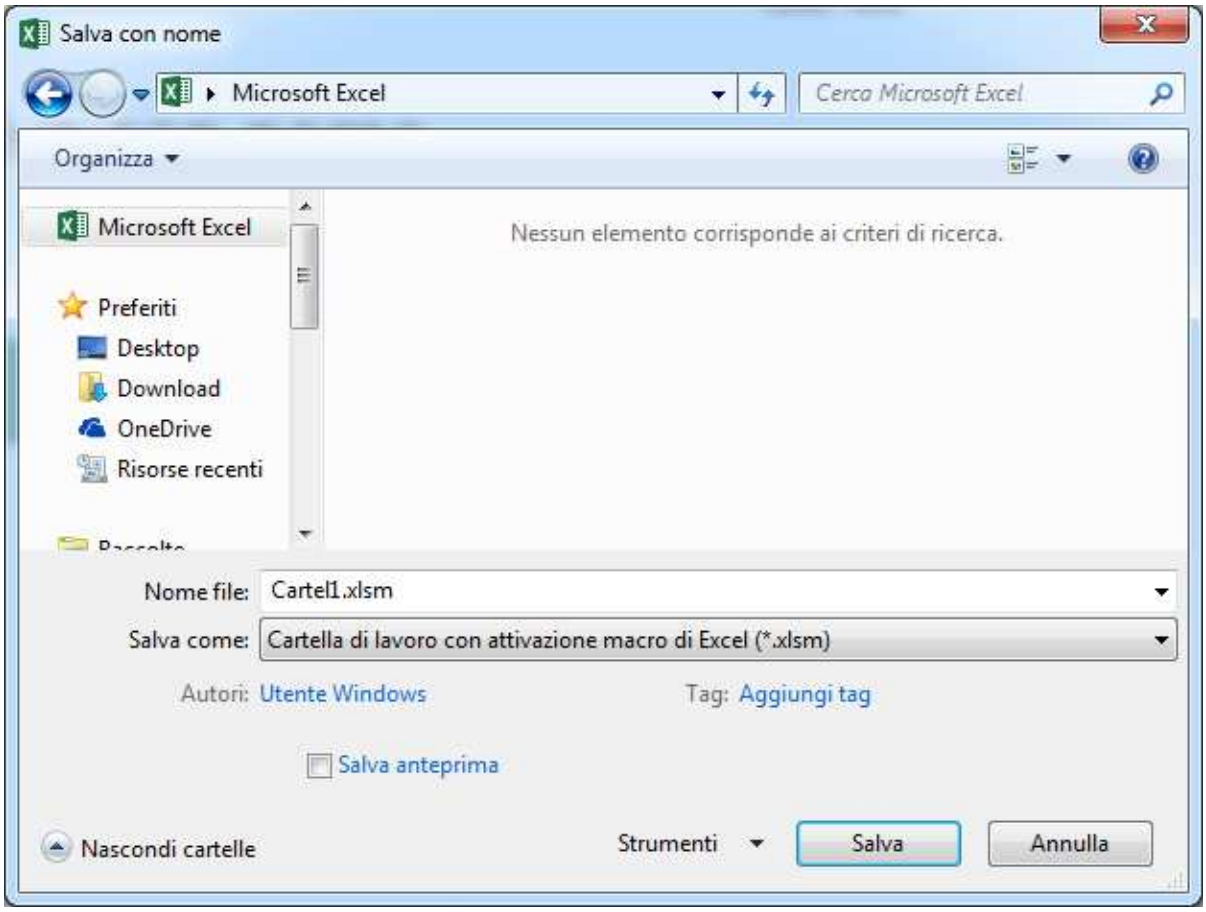


Fig. 2

Qualora si tenti di salvare il file nel formato tradizionale, Excel avviserebbe che non è possibile salvare le caratteristiche del vostro file nel formato scelto

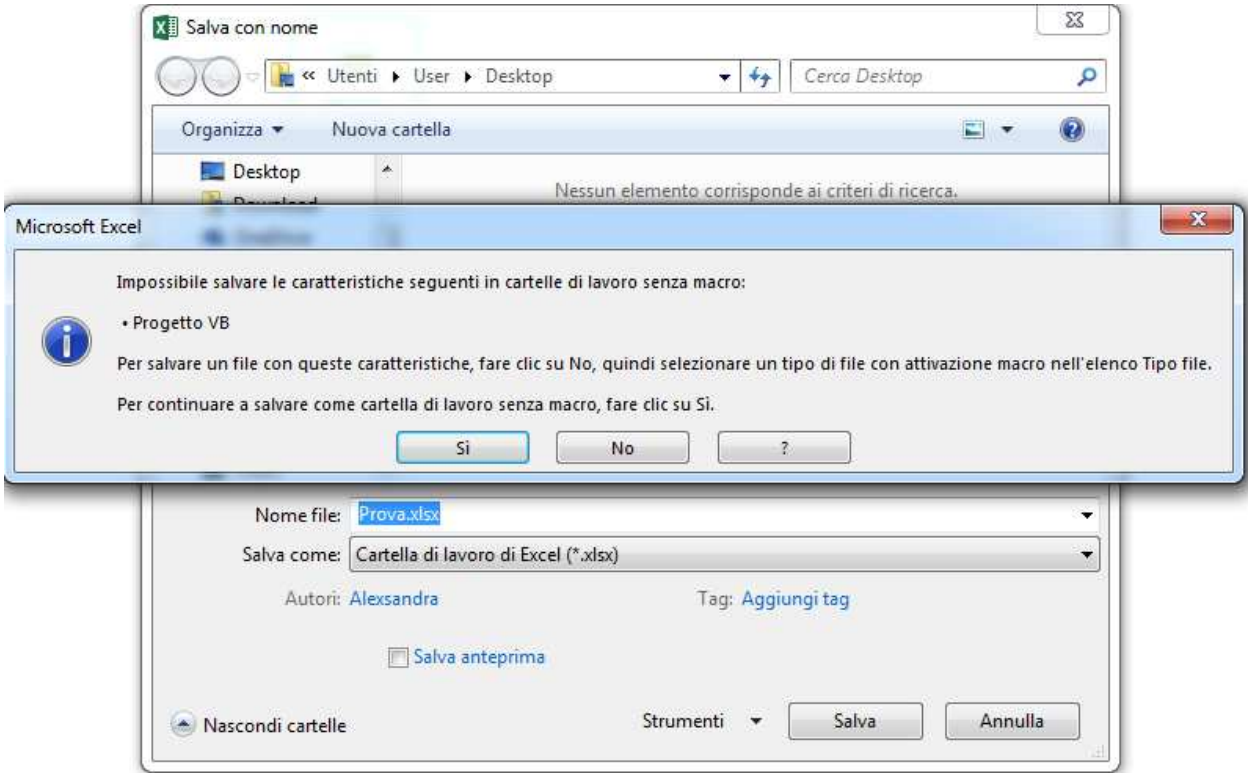


Fig. 3

Nella finestra di avviso sopra riportata si deve scegliere **No**, in questo modo Excel mostrerà la finestra *Salva con nome*, nella quale è possibile scegliere il formato corretto per conservare il progetto VBA. Se, invece, viene scelta l'opzione **Sì** e si procede con il salvataggio comunque in formato **.xlsx**, il progetto VBA inserito nel file non sarà in alcun modo eseguibile né utilizzabile.

NB *Nelle versioni di Office XP e Office 2003 non vengono distinti i file con codice VBA da quelli che ne sono privi e tutti i file Excel hanno la stessa estensione (**.xls**) e la stessa icona. La differenziazione del formato dei file che contengono codice VBA si è resa necessaria anche per il fatto che il codice VBA può essere pericoloso. L'esecuzione di codice che, di fatto, compie delle operazioni all'insaputa dell'utente, può comportare dei rischi e, non di rado, si sono trovati documenti Office che, all'interno del VBA, nascondevano dei virus.*

Per questo, oltre a differenziare il formato, Excel permette di stabilire come gestire i singoli file che contengono codice. Volendo, è possibile fare in modo che Excel blocchi l'esecuzione di tutto il codice, ma, avendo bisogno del VBA, si tratta di una impostazione troppo drastica. D'altro canto, lasciare che Excel esegua qualsiasi codice, anche quello non scritto da voi, può essere pericoloso. La soluzione giusta consiste nel fare in modo che, all'apertura di tutti i file che contengono codice, Excel chieda se eseguirlo o meno.

Per determinare queste impostazioni occorre che, sulla barra multifunzione, sia visibile la scheda **Sviluppo**, se non lo fosse, si deve seguire il percorso **File - Opzioni** e una volta visualizzata la finestra delle opzioni andare alla scheda **Personalizzazione barra multifunzione** e nel riquadro di destra, selezionare la scheda **Sviluppo** e poi premete **Ok** per tornare al foglio di lavoro.

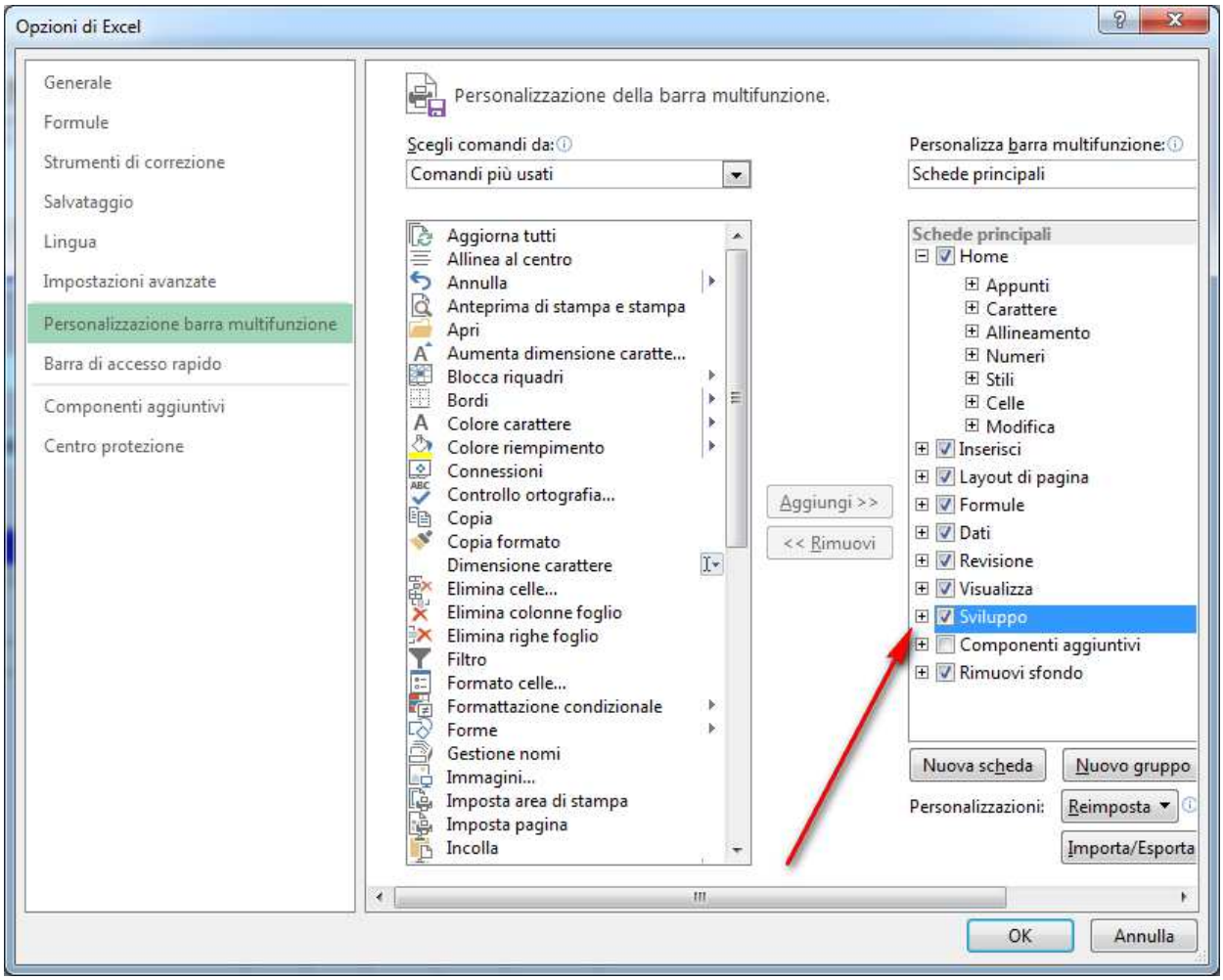
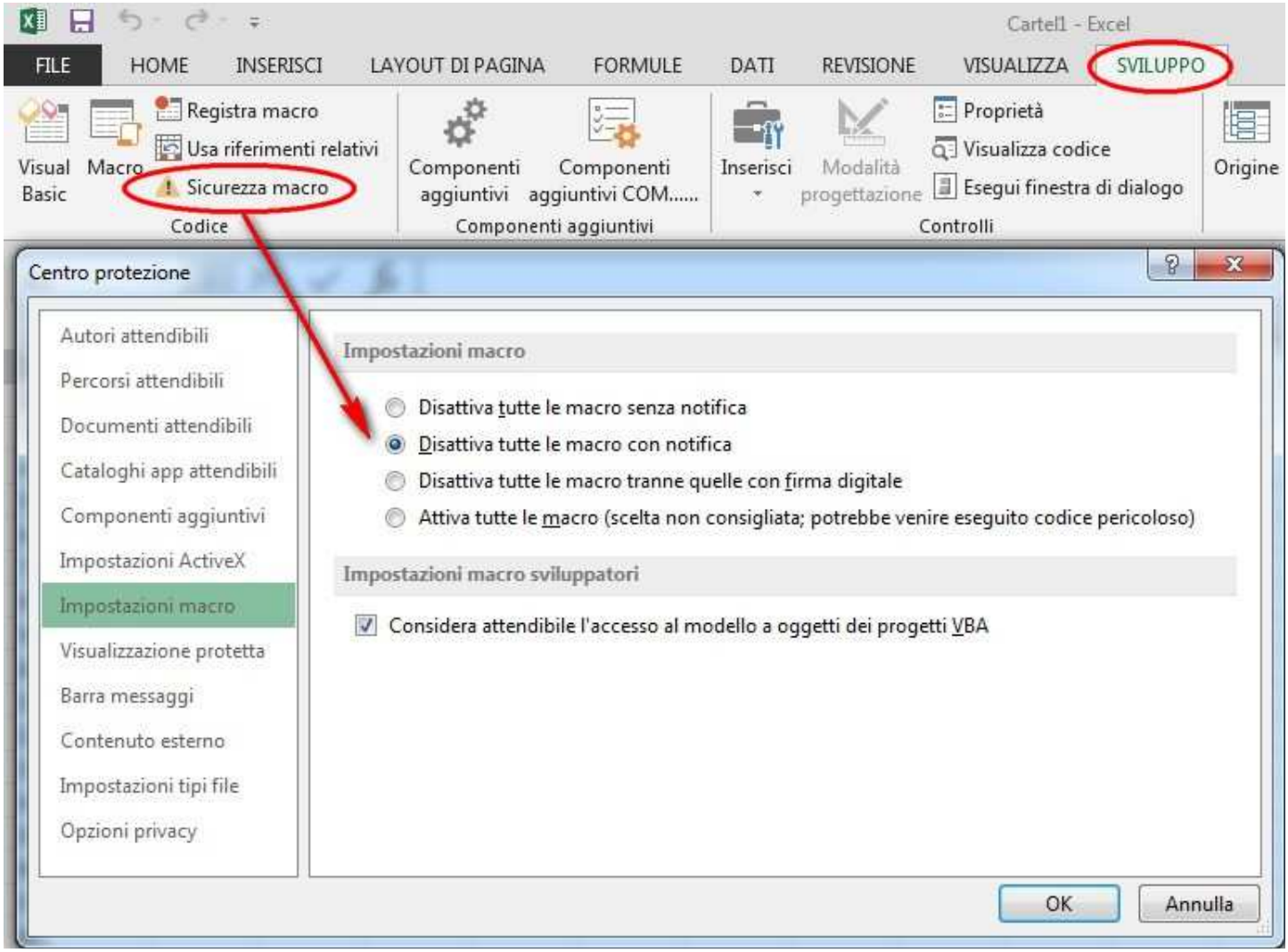


Fig. 4



NB

Chi utilizza Excel 2007, per visualizzare la scheda Sviluppo, deve aprire il menu del pulsante Microsoft Office e premere il pulsante Opzioni di Excel, comparirà la finestra Opzioni di Excel e nella sezione Impostazioni generali occorre mettere un segno di spunta alla voce Mostra scheda Sviluppo sulla barra multifunzione.

Introduzione agli Oggetti Excel

La programmazione VBA è utilizzata per ottenere una migliore funzionalità di calcolo, per automatizzare le operazioni ripetitive e per integrare Excel con altre applicazioni di Office. Le istruzioni date a Excel sono sotto forma di codice, e vengono chiamate macro o procedure e sono sviluppate in **Visual Basic Editor** (VBE), che è l'ambiente di sviluppo VBA. Utilizzando il **VBA** è possibile facilitare le operazioni necessarie per ottenere un risultato, sia esso di calcolo che di ricerca, attraverso delle azioni svolte su un *Oggetto, i*n pratica, attraverso un semplice *Click* su un pulsante, il computer elabora una serie di dati, agevolando così il nostro lavoro.
È necessario, per prima cosa, illustrare alcune piccole terminologie di base che identificano le varie fasi della programmazione, ricordando che ogni oggetto, o controllo, presente nella casella degli strumenti è dotato di un insieme di **proprietà**, **metodi** ed **eventi**.

Oggetti, Proprietà e Metodi

Un oggetto è una "cosa" che contiene i dati e ha ***proprietà*** e ***metodi*** e sono una parte fondamentale della programmazione VBA. Gli **oggetti** sono la cartella di lavoro, il foglio di lavoro, un pulsante di comando, i font, i pulsanti di opzione che andiamo ad inserire su un foglio e ai quali affidiamo il compito di eseguire una determinata azione

- **Le Proprietà s**ono le caratteristiche o gli attributi che descrivono l'oggetto, come il nome, il colore, la dimensione, e definiscono il comportamento di un oggetto, se è visibile o abilitato. I dati o le informazioni di un oggetto possono essere raggiunti con proprietà o metodi, dove il metodo è un'azione eseguita da un oggetto tramite un codice VBA che farà sì che l'oggetto esegua un'azione.

- **I metodi** sono le istruzioni, impartite ad un oggetto, relative alle azioni che deve intraprendere in risposta ad un evento. Ogni tipo di oggetto ha un elenco di metodi diversi e specifici, per esempio una Userform può compiere azioni come: *Unload* (il form si cancella dalla memoria temporanea del computer per lasciare il posto ad un altro form o per terminare il programma), *Hide* (il form si nasconde durante l'esecuzione del programma), *Show* (il form si mostra in un momento determinato, durante l'esecuzione del programma), etc.

- **Gli Eventi s**ono le azioni, che vengono svolte durante l'utilizzo del programma dall'utente finale e sono connesse ad un oggetto. Una routine evento (un codice VBA) viene attivata quando si verifica un evento come l'apertura, la chiusura, l'attivazione o disattivazione della cartella di lavoro, la selezione di una cella o cambiando la selezione delle celle in un foglio di lavoro, fare un cambiamento nel contenuto di un foglio di lavoro, la selezione o l'attivazione di un foglio di lavoro, e così via. Excel dispone di routine evento che sono procedure richiamate automaticamente quando un oggetto riconosce il verificarsi di un evento.

Le procedure di evento sono collegate a oggetti come la cartella di lavoro, il foglio di lavoro, i grafici, le Form o i vari controlli e sono attivate da un evento predefinito. Vengono installate all'interno di Excel con un nome standard e predeterminato, come la procedura di modifica del foglio di lavoro viene installata con il foglio di lavoro e denominata "*Private Sub Worksheet_Change*". In questa routine l'oggetto foglio di lavoro è associato all'evento *Change*, che contiene un codice personalizzato che viene eseguito automaticamente quando si modifica il contenuto di una cella del foglio di lavoro.

Il modello a oggetti ***Application*** rappresenta l'intera applicazione Excel e contiene tutti gli oggetti di programmazione legati gli uni agli altri in una gerarchia. L'oggetto Application è in cima alla gerarchia di oggetti di Excel e spostandosi verso il basso è possibile accedere all'oggetto cartella di lavoro (Workbook), ai fogli di lavoro (Wotsheet), alle Celle etc.

Gli oggetti di Excel sono accessibili attraverso oggetti "*padri*", il foglio di lavoro è il genitore dell'oggetto Range (Celle), la cartella di lavoro (Workbook) è il padre dell'oggetto foglio di lavoro (Worksheet), e l'oggetto Application è il padre dell'oggetto Workbook.

Prima di inoltraci ulteriormente negli argomenti è opportuno almeno conoscere il significato di certi termini e aspetti della programmazione VBA. In questa fase indicheremo in maniera marginale il significato e la sintassi di certi argomenti, che verranno ampiamente trattati nel proseguo del corso in maniera più chiara ed esauriente.

L'insieme Workbooks

Come abbiamo poco sopra affermato l'oggetto Workbook rappresenta una cartella di lavoro di Microsoft Excel, ed è un elemento dell'insieme Workbooks che contiene tutti gli oggetti Workbook aperti in Microsoft Excel. Vediamo alcuni esempi di metodi e proprietà per l'oggetto Workbook.

*Workbooks.*Close == > [chiude tutte le cartelle di lavoro aperte]
Workbooks.Add == > [aggiunge una nuova cartella di lavoro vuota]
Workbooks(1).Activate == > [attiva la prima cartella di lavoro aperta]

Il numero di indice determina l'ordine nel quale le cartelle di lavoro sono state aperte o create, per cui: *Workbooks(1)* rappresenta la prima cartella di lavoro creata

L'insieme Worksheets

L'insieme Worksheets è un insieme di tutti gli oggetti Worksheet nella cartella di lavoro specificata o attiva e ciascun oggetto Worksheet rappresenta un foglio di lavoro.

Worksheets(1) == > [è il primo foglio di lavoro da sinistra della cartella di lavoro]
Worksheets(1).Visible = False == > [nasconde il foglio 1 della cartella di lavoro attiva]
Worksheets("Foglio1").Activate == > [attiva Foglio1]

Quando il foglio attivo è un foglio di lavoro, per farvi riferimento è possibile utilizzare la proprietà "*ActiveSheet*"

ActiveSheet.Name = "Archivio"
MsgBox ActiveSheet.Name == > [riporta il nome del foglio attivo]
ActiveSheet.PrintOut == > [stampa il foglio di lavoro]

L'insieme Sheets

Sheets è l'insieme di tutti i fogli della cartella di lavoro specificata o attiva e può contenere l'oggetto Chart (grafico) o Worksheet (foglio di lavoro).

Sheets.PrintOut == > [stampa tutti i fogli della cartella di lavoro attiva]
Sheets("Foglio1").Activate == > [attiva Foglio1]

L'insieme Range

Rappresenta una cella, una riga, una colonna, una selezione di celle contenente uno o più blocchi contigui di celle.
Per restituire un oggetto Range che rappresenta una singola cella o un intervallo di celle, la modalità di utilizzo è

Worksheets("Foglio1").Range("A5").Value = 15 == > [Inserisce il valore 15 nella cella A5]

Quando viene utilizzata senza qualificatore di oggetto (vale a dire l'oggetto prima del punto), la proprietà Range restituisce un intervallo del foglio attivo. Si può utilizzare il metodo Activate per attivare un foglio di lavoro prima di utilizzare la proprietà Range senza il qualificatore di oggetto

Worksheets("Foglio1").Activate
Range("A1:A10").Formula = "=Rand()" == > [inseriamo un numero causale nell'intervallo specificato]

La Proprietà Cells

La proprietà Cells restituisce un oggetto che rappresenta tutte le celle del foglio di lavoro attivo e la modalità di utilizzo è Cells(row, column) dove row è l'indice di riga e column è l'indice della colonna, per restituire una singola cella.

Worksheets(1).Cells(1, 1).Value = 15 == > [imposta il valore della cella A1 a 15]
ActiveSheet.Cells(2, 1).Formula = "=Somma(A1:B10)" == > [imposta la formula nella cella A2]

La Proprietà Columns

Per un oggetto Application restituisce un oggetto che rappresenta tutte le colonne del foglio di lavoro attivo mentre per un oggetto Range restituisce le righe dell'intervallo specificato. Utilizzare questa proprietà senza un qualificatore di oggetto equivale ad ActiveSheet.Columns.

Worksheets("Foglio1").Columns(1).Font.Bold = True == > [imposta il carattere grassetto alla Colonna A di Foglio1]
Columns(1).Value = 0 == > [imposta il valore di tutte le celle a zero della colonna 1]

La Proprietà Rows

Per un oggetto Application, restituisce un oggetto che rappresenta tutte le righe del foglio di lavoro attivo mentre per un oggetto Worksheet, restituisce tutte le righe del foglio di lavoro specificato. Utilizzare questa proprietà senza un qualificatore di oggetto equivale a ActiveSheet.Rows.

Worksheets("Foglio1").Rows(3).Delete == > [elimina la riga 3 di Foglio1]

Oggetti e Routine in VBA

Abbiamo visto che il VBA è un linguaggio di programmazione *orientato agli oggetti* e consente di gestire le applicazioni con semplicità in quanto *la Programmazione ad Oggetti* è un sistema di organizzazione del codice di un programma mediante raggruppamento all'interno di oggetti, ovvero singoli elementi che presentano *Proprietà* e *Metodi*. In pratica non è più il programma a gestire il flusso di istruzioni, ma sono gli oggetti che interagiscono tra di loro intervenendo sulle azioni del programma.

Riferendoci alla nostra cartella di lavoro possiamo dire che essa è costituita da oggetti di varia natura, come gli oggetti grafici o i controlli (finestre di dialogo, pulsanti ecc.) incollati su un foglio, ma non è questa la peculiarità di un foglio elettronico la cui ossatura è costituita, partendo dal livello più basso, da:

- celle
- intervalli
- fogli
- cartelle di lavoro

Ricordiamo inoltre che se osserviamo questo insieme di oggetti come un albero genealogico il capostipite, o il culmine, spetta all'oggetto, denominato *Application*, che rappresenta, tutti gli Oggetti di Excel. Si tratta appunto di Excel stesso. Proseguendo nella discesa genealogica troveremo i seguenti oggetti:

- L'oggetto Workbook : Che è la cartella di lavoro (cioè il nostro file)
- L'oggetto Worksheet : Che è il foglio di lavoro (Foglio1, Foglio2 ecc...)
- L'oggetto Range : Che è un intervallo di celle (A1: B12, C1:D12, ecc...)

Sintatticamente possiamo affermare che ciascun oggetto fa parte di una famiglia o classe e l'accesso al singolo membro di ciascuna classe si effettua attraverso metodi, *pluralistici*, cioè da una pluralità di metodi che collaborano e interagiscono con i vari oggetti in maniera omogenea e ai quali corrispondono insiemi di oggetti quali :*Workbooks*, *Worksheet*, *Range* e *Cells*, inoltre i membri più elevati si possono omettere nel caso che il soggetto sia attivo; vedremo meglio questo passaggio fra poche righe.

Fin qui abbiamo delineato i componenti principali cercando di esporre come vengono interpretati dal Visual Basic applicato al foglio elettronico, ma l'obiettivo vero è quello di focalizzare gli oggetti che formano l'ossatura, il nucleo di un foglio di lavoro, al cui centro, vi sono intervalli e celle con il loro contenuto di dati da elaborare o formule e il loro inquadramento nel mondo Visual Basic è fondamentale e aiuterà a capire meglio tutto il resto, vediamo di interpretare quanto appena affermato usando il Visual Basic. Con queste sintassi

Workbooks("prova.xls") e *Worksheets("Foglio1")*

Si individuano rispettivamente la cartella e il foglio di lavoro (notare i loro nomi virgolettati dentro le parentesi), ma dobbiamo però tenere presente che un elemento può anche venire individuato tramite un indice, il quale può essere o il numero di ordine o il nome fra virgolette, per capire meglio il concetto di indice possiamo dire che la sintassi Workbooks(2) e Workbooks("prova.xls") puntano entrambe alla stessa cartella prva.xls a patto che questa sia la **seconda** fra quelle aperte contemporaneamente da Excel.

Infatti se abbiamo solo la cartella prova.xls aperta, la sintassi esatta diventa *Workbooks(1)*. A questo punto è abbastanza chiaro che l'indice che usiamo fra parentesi nell'oggetto Workbooks varia ed è strettamente legato al numero di cartelle aperte nel momento dell'esecuzione di questa istruzione, pertanto possiamo far notare che è possibile scrivere questa istruzione in tre diversi modi:

- *Application.Workbooks(1).Worksheets(1).Range("A1:B 10")*
- *Application.Workbooks("prova.xls").Worksheets(1).Range("A1:B10")*
- *Application.Workbooks("prova.xls").Worksheets(" Foglio1").Range("A1:B10")*

Ricordate che poco sopra abbiamo però affermato che i membri più elevati si possono omettere quando sono attivi, per cui:

- Se abbiamo Excel aperto
- Se la cartella prova.xls è aperta
- Se infine ci troviamo nel Foglio1

Possiamo ridurre tutto il listato ad un semplice Range("A1:B10"), in caso contrario credo che sia abbastanza chiaro come agire usando gli indici per individuare il nostro intervallo (o Range). Si deve inoltre prestare attenzione alle proprietà "pluralistiche" che poco sopra abbiamo citato: le prime volte è facile scordarsi del plurale, scrivendo *Workbook*("prova.xls") anziché Workbooks("prova.xls") oppure *Worksheet*("Foglio1") invece di Worksheets("Foglio1").

Le Routine

Finora abbiamo usato diverse volte il termine **Routine**, vediamo ora di capire cosa sia. Una routine (o procedura) è definita come un insieme di codici che permettono di eseguire un'azione ed è generalmente di due tipi: *Public* o *Private*.

- *Public* è quando ha valore per l'intero progetto
- *Private* ha validità solo all'interno del *Modulo* sul quale e stata scritta.

Una routine inizia con la dichiarazione della parola chiave ***Sub***, seguita da un nome e poi seguita da una serie di parentesi e si conclude con una dichiarazione ***End Sub*** che può essere digitato, ma appare automaticamente dopo aver digitato il nome della procedura seguito dalla pressione del tasto Invio per andare alla riga successiva. Il codice VBA o le dichiarazioni sono inseriti in mezzo tra le due dichiarazioni

Public Sub CommandButton1_Click()
altro Codice ……..
End Sub

Come eseguire una routine

Ci sono molti modi per eseguire una macro, possiamo utilizzare la finestra di dialogo macro, o il tasto di scelta rapida, o in VBE o assegnando la macro a un oggetto. La finestra di dialogo macro è raggiungibile dalla scheda *Visualizza* della barra multifunzione, cliccando su ***macro*** e poi su ***Visualizza macro*** che aprirà la finestra di dialogo macro, in cui si deve selezionare il nome della macro e cliccare sul pulsante ***Esegui*** per eseguire la Sub/macro.

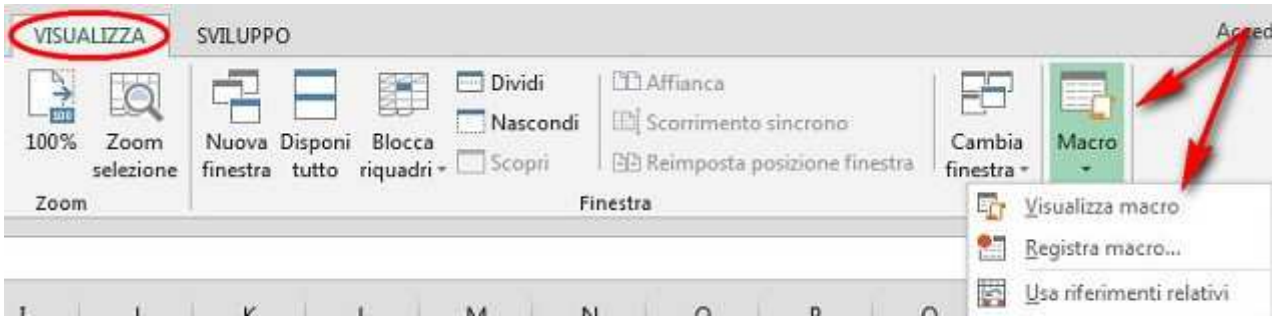


Fig. 1

Fig. 2

È inoltre possibile aprire la finestra di dialogo macro facendo clic su ***Macro*** nel gruppo Codice della scheda Sviluppo sulla barra multifunzione.

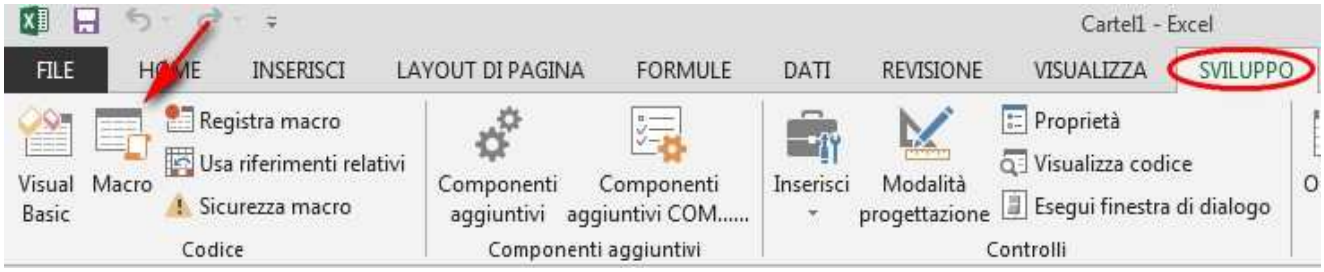


Fig. 3

E' inoltre possibile eseguire una macro direttamente dall'editor di ***Visual Basic***

- In Visual Basic Editor, si deve fare clic all'interno della sub e premere F5 (o fare clic su Esegui nel barra dei menu e quindi fare clic su *Esegui* Sub/UserForm).
- Oppure, sempre in Visual Basic Editor, si deve fare clic su *Strumenti* nella barra dei menu, quindi fare clic su *macro* che apre la finestra di dialogo Macro (Fig. 2). Nella finestra di dialogo Macro, selezionare il nome della macro e fare clic su *Esegui* per eseguire la macro.

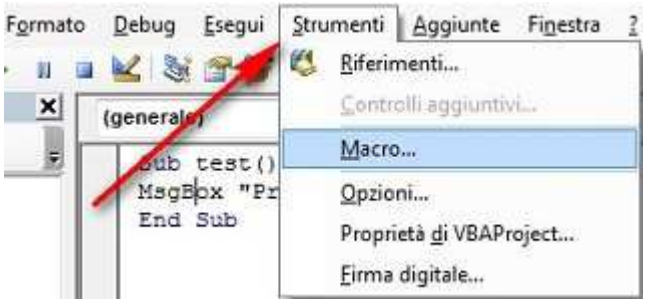


Fig. 4

Un buon modo per eseguire una routine (o macro) potrebbe essere quella di fare clic su un pulsante accompagnato da un testo esplicativo posto sul foglio di lavoro, per questo è necessario assegnare la macro a un oggetto, forma, grafico o controllo. È possibile assegnare una macro a qualsiasi controllo agendo dalla scheda **Sviluppo** sulla barra multifunzione, cliccando su ***Inserisci*** nel gruppo *Controlli*, selezionare e fare clic sul pulsante nei controlli modulo e quindi fare clic sul foglio di lavoro in cui si desidera posizionarlo nell'angolo superiore sinistro del pulsante (è possibile spostare e ridimensionare in seguito).

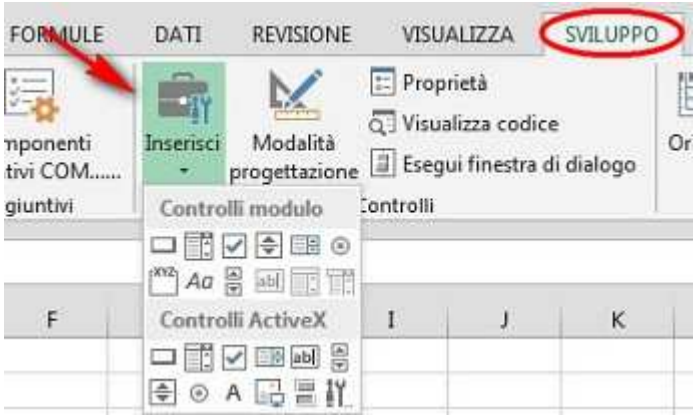


Fig. 5

Successivamente cliccando col destro del mouse sul pulsante e dal menu a discesa che compare selezionare ***Assegna macro*** e si apre la finestra di dialogo *Assegna macro* dal quale è possibile selezionare e assegnare una macro all'oggetto disegnato.

Fig. 6

E' possibile assegnare macro a qualsiasi oggetto, forma, immagine, grafico etc. inserito nel foglio di lavoro. Nella scheda Inserisci della barra multifunzione, è possibile inserire un oggetto, forma, immagine, elemento grafico, Tabella, Casella di testo, WordArt, etc. nel foglio di lavoro.

Prefazione Editor VBA

Di solito si utilizza Microsoft Excel per creare documenti normali utilizzando le impostazioni predefinite dell'applicazione, ma se vogliamo applicare delle funzioni avanzate ad un foglio di calcolo, possiamo utilizzare Microsoft Visual Basic, un'applicazione che si installa automaticamente con Microsoft Excel.

Per creare un foglio di calcolo con funzionalità avanzate, si deve scrivere del codice nell'ambiente di programmazione Visual Basic Applications Edition (VBA). Per sfruttare le funzionalità dell'ambiente VBA, ci sono molte regole da seguire per interagire con il linguaggio

Visual Basic Editor (VBE)

VBE è contenuto nella cartella di lavoro di Microsoft Excel, ed è un ambiente usato per scrivere, modificare ed eseguire il debug di codice VBA che si può avviare in Excel 2007 dai seguenti percorsi:

- Dalla scheda **Sviluppo - Visual Basic**
- Dalla scheda **Sviluppo - Visualizza codice**
- Fare clic col destro del mouse sulla scheda del nome foglio in basso, quindi fare clic su **Visualizza codice**
- Premere la combinazione di tasti **Alt + F11**

I componenti di Visual Basic Editor si riferiscono alla *Finestra del Codice*, *la Finestra Gestione progetti*, *la finestra Proprietà* e l'area di lavoro di programmazione (es. menu e barre degli strumenti, oggetti, la finestra Immediata e la finestra di controllo). La Finestra del codice è dove scrivere e modificare il codice e le procedure, e anche dove le macro vengono registrate, mentre esplora progetti visualizza l'elenco di tutti i progetti esistenti e fornisce una vista ad albero in cui è possibile comprimere, per nascondere, o espandere, per visualizzare, gli oggetti, i Form e i moduli contenuti in un progetto. Ogni progetto contiene una cartella oggetti, che è il *Microsoft Excel Objects*, una cartella Forms che contiene i Form, una cartella Moduli che contiene i moduli e la cartella moduli di classe che contiene le classi.

La cartella Objects è sempre presente e contiene un oggetto foglio per ogni foglio di lavoro esistente, e un oggetto il *ThisWorkbook* e ogni oggetto foglio ha un primo nome che appare prima delle due parentesi, che è il nome del foglio e un secondo nome che compare tra le parentesi che è il nome della scheda del foglio che appare nel foglio di lavoro di Excel. Per visualizzare la finestra di Esplora progetti, si deve cliccare su *Visualizza* sulla barra dei menu VBE e quindi selezionare *Esplora progetti*, o premere CTRL + R in VBE.

Moduli in Excel VBE

Le macro VBA (cioè i codici o le procedure) devono risiedere nei loro appositi moduli o Excel non sarà in grado di trovare ed eseguirli, l'oggetto Modulo viene utilizzato per le procedure di evento incorporate in Excel e per creare i propri eventi che sono: Modulo, *ThisWorkbook*, moduli Sheet (fogli di lavoro e fogli grafici), moduli Form e moduli di classe, in generale il codice VBA nei confronti degli eventi non associati a un particolare oggetto (come la cartella di lavoro o un foglio) vengono inseriti in un modulo di codice standard, una pratica generalmente accettata è quella di piazzare le routine di evento nel modulo ThisWorkbook, nei moduli di fogli e UserForm, pur ponendo altro codice VBA in moduli standard. I moduli di codice standard sono anche indicati come moduli di codice o moduli, e ci possono essere vari moduli (Modulo1, Modulo2 etc.) in un progetto VBA, in cui ciascun modulo può essere utilizzato per coprire un certo aspetto del progetto. Per inserire un modulo si deve cliccare su *Inserisci* sulla barra dei menu VBE e quindi selezionare *Modulo*.

Proseguimento linea del codice VBA

Molte volte durante la scrittura di codice VBA, una sola riga di codice potrebbe diventare molto lunga e nella finestra del codice sarà necessario scorrere verso il lato destro per leggerlo. Per dividere una singola riga in più righe in VBA, è possibile utilizzare la linea come carattere di continuazione che è la combinazione di uno spazio seguito da un underscore (_), usando questo carattere come una interruzione di linea, vi permetterà di passare alla riga successiva e così via, e questo sarà il trattamento di tutte le linee continue come una singola riga nel codice VBA.

Si noti che si può avere un massimo di 25 linee fisicamente, unite con il carattere di continuazione di riga (cioè un massimo di 24 caratteri di continuazione della riga), e di essere trattate come una singola riga logica di codice. Una linea fisica può avere al massimo 1023 caratteri mentre una linea logica può avere un massimo di 10.230 caratteri, in modo che un massimo di 25 linee fisiche possono essere unite pari ad un massimo di 10.230 caratteri.

Controllo Automatico Sintassi

Durante la scrittura di codice VBA, il codice che ha un errore di sintassi diventa rosso non appena il cursore si sposta sulla linea se il controllo automatico di sintassi è selezionato (impostazione di default), un controllo della sintassi viene eseguito ogni volta che si sposta il cursore su una nuova linea, e in caso di errore di sintassi una finestra pop-up avverte con un messaggio "*errore di compilazione*". Se il controllo automatico di sintassi è deselezionato, non sarà più possibile ottenere questi box di avviso, anche se l'errore farà diventare di colore rosso la riga del codice. È possibile deselezionare il controllo automatico di sintassi andando in VBE, seguendo il percorso ***Strumenti – Opzioni*** e nella finestra visualizzata, selezionare la scheda Editor, e quindi deselezionare la casella di controllo.

È inoltre possibile modificare il colore rosso di default dell'errore di sintassi andando in VBE, cliccate su ***Strumenti – Opzioni*** e nella finestra visualizzata, selezionare la scheda Formato, selezionare sintassi del testo di errore nella casella di riepilogo codice colori, e poi scegliere il nuovo colore di primo piano.

Commento del codice in VBA

All'interno di una procedura, durante la scrittura di codice è possibile contemporaneamente fornire commenti per spiegare lo scopo e ciò che il codice sta facendo. Per differenziare i commenti con il codice, dovete inserire un singolo carattere di apostrofo (') o con la dicitura **Rem** seguito da uno spazio. I commenti verranno ignorati da VBA in modo che un errore di sintassi nel commento non viene restituito. Si consiglia di utilizzare i commenti nel codice che saranno di grande aiuto a un altro utente nella comprensione, o se si deve effettuare una modifica in un secondo momento. Una volta che si aggiunge un carattere di commento (') all'inizio di una riga è possibile utilizzare la sequenza di continuazione (_) per continuare il commento alla riga successiva e quando si preme il tasto Invio dopo aver digitato un commento, il suo colore diventa verde, come da impostazione predefinita di VBA.

Questa impostazione predefinita del colore del commento può essere modificato andando in VBE, e seguendo il percorso ***Strumenti – Opzioni***

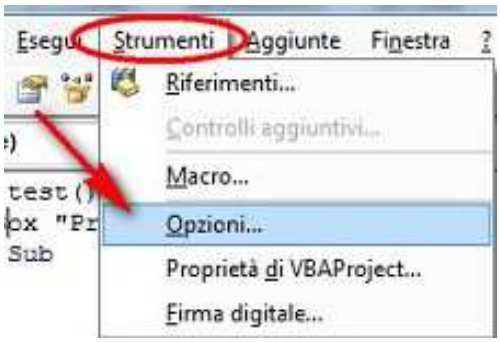


Fig. 1

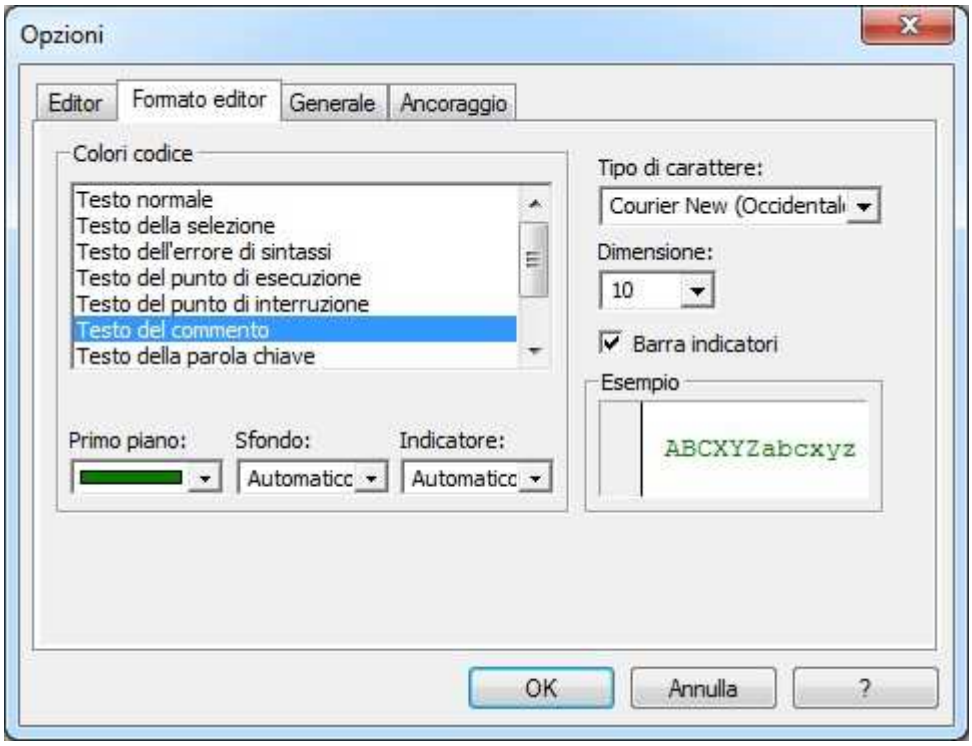


Fig. 2

Indentare il codice VBA

La procedura di VBA può essere composto da più righe di codice con una serie di dichiarazioni e quando il codice diventa più lungo e complesso, la formattazione del codice con un rientro contribuirà a rendere più facile da leggere, e renderà il debug più semplice. Gli sviluppatori utilizzano in genere un rientro per il codice all'interno delle linee di inizio e fine dei Loop in cui una serie di righe di codice o in un ciclo For Next si usa rientrare per distinguerli come appartenenti ad un particolare procedimento. L'indentazione è molto utile nel codice nidificato in modo che ogni blocco di codice è visivamente separato, e le linee di inizio e fine di ogni blocco sono allineate.

L'Indentazione di solito è fatta premendo il tasto Tab una o più volte, prima di digitare il codice. Si noti che per impostazione predefinita in VBA, premendo il tasto Tab si sposta il cursore di quattro spazi o caratteri a destra. Questa impostazione del valore della scheda può essere modificata andando a VBE, e seguendo il percorso ***Strumenti – Opzioni*** e nella finestra visualizzata selezionare la scheda Editor e inserire il nuovo valore nella casella "*Larghezza linguetta*". Nella scheda Editor della finestra di dialogo Opzioni, è possibile anche selezionare "*Rientro tabulazione*", che ripeterà il rientro della riga corrente premendo Invio.

Parole chiave di VBA

Le parole chiave sono parole riservate che VBA usa come parte del suo linguaggio di programmazione e sono parole o comandi che sono riconosciuti da VBA e possono essere utilizzati nel codice solo come parte del linguaggio VBA (come in una dichiarazione, il nome della funzione, o l'operatore) e non altrimenti (come i nomi delle sub-routine o variabile). Esempi di parole chiave sono: Sub, End, Dim, If, Next, And, Or, Loop, Do, Len, Close, data, Else, Select, e così via. Per ottenere aiuto su una determinata parola chiave, inserire il cursore del mouse all'interno della parola (nel codice VBA in VBE) e premere F1.

Inserire un Modulo nell'Editor VBA

Per inserire una macro si deve accedere all'editor VBA cliccando sull'icona di Visual Basic nella scheda **Sviluppo** della barra multifunzione, oppure è possibile premere i tasti **ALT** + **F11** sulla tastiera

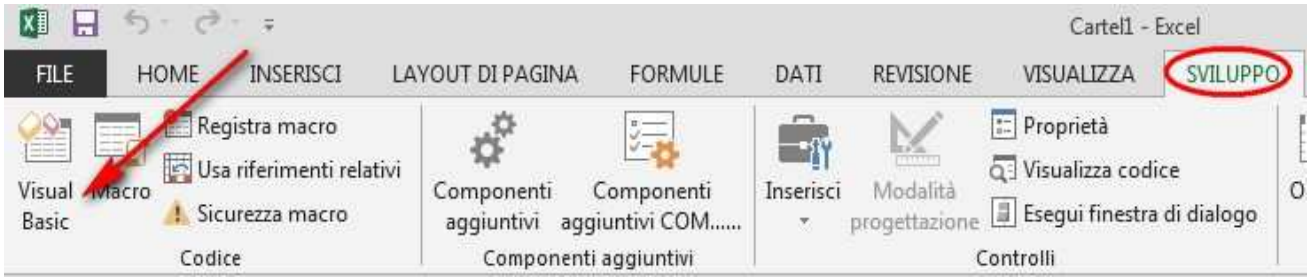


Fig. 1

Le macro sono memorizzate in moduli, guardate nella finestra del progetto VBA e se la finestra non fosse visibile si deve seguire il percorso "**Visualizza – Gestione Progetti**" dal menu. Selezionare poi la cartella di lavoro in cui si desidera inserire il codice, che si possono identificare dal nome della cartella tra parentesi accanto alla dicitura "**VBAProject**". In questo modo ogni cartella di lavoro aperta avrà una voce nella finestra del progetto VBA

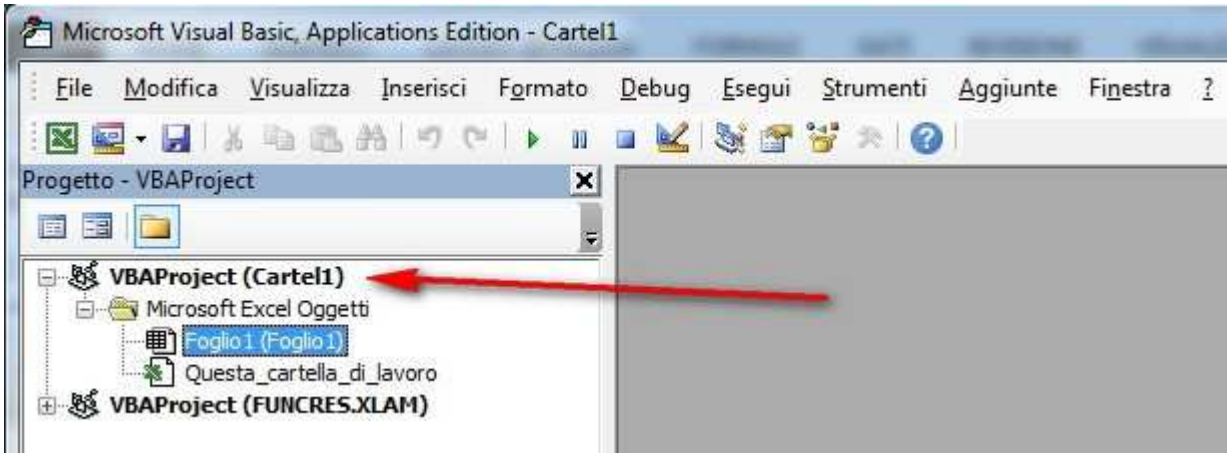


Fig. 2

A questo punto si deve fare clic col pulsante destro del mouse sulla cartella di lavoro in cui si desidera inserire il codice e selezionare "**Inserisci**" e poi "**Modulo**" dal menu che appare

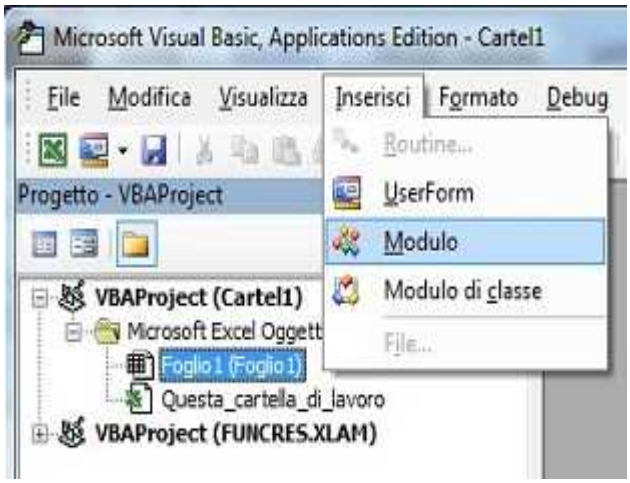


Fig. 3

E verrà creato un nuovo modulo:

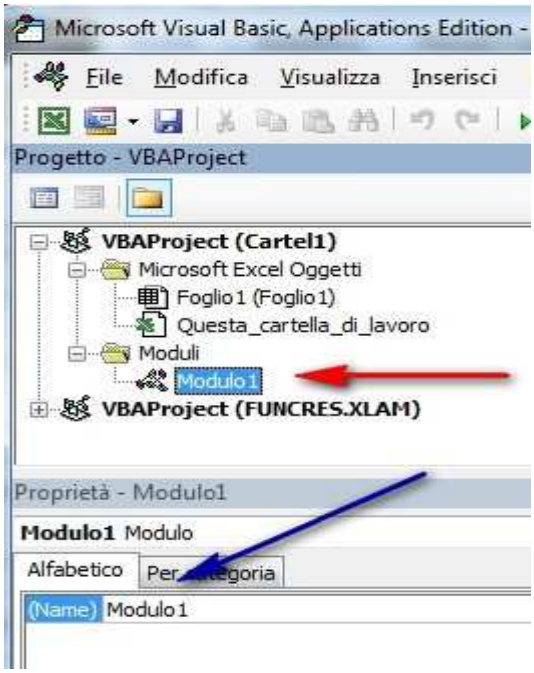


Fig. 4

Quando si fa doppio clic su un modulo nella finestra di progetto VBA il codice di tale modulo viene visualizzato nella finestra del codice principale, anche se un nuovo modulo normalmente non contiene alcun codice (può contenere "Option Explicit" se si dispone di questa opzione attivata). E' possibile modificare il nome del modulo cliccando sul nome stesso nella finestra delle proprietà, evidenziata dalla freccia blu in Fig. 4

Creare una macro

Per creare una macro si deve cliccare sul modulo appena creato e nella finestra del codice digitare il nome assegnato alla macro (prova) preceduto dalla notazione **Sub** e premere. Si dovrebbe ottenere una cosa del genere

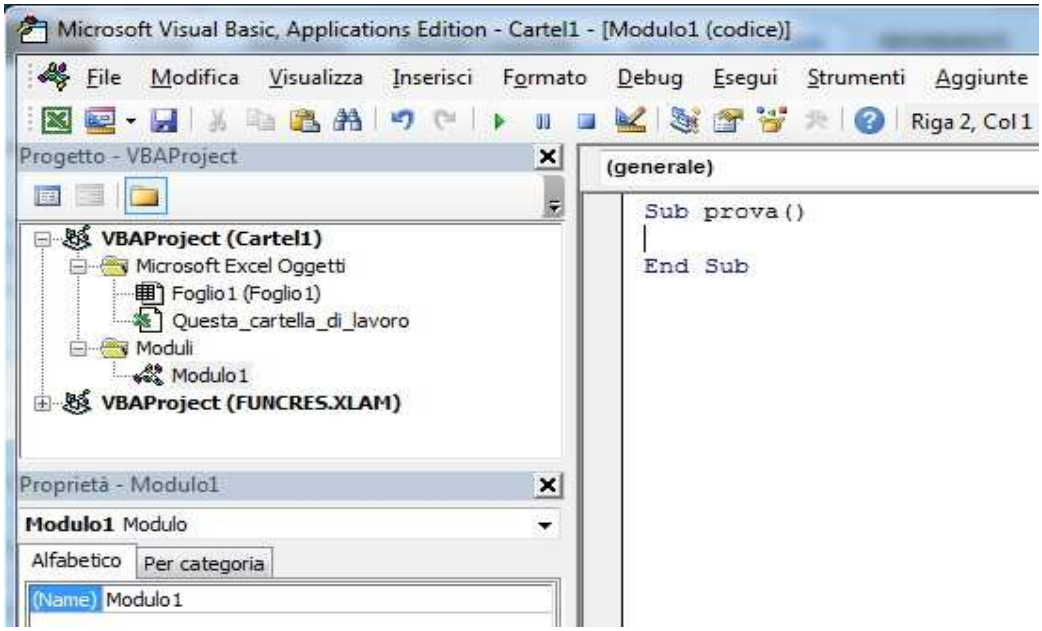


Fig. 5

A questo punto alla macro si deve solo aggiungere il codice per svolgere le azioni che vogliamo. Per esempio possiamo far apparire un box a video con un saluto, in questo caso aggiungiamo il codice:

MsgBox “Salve”

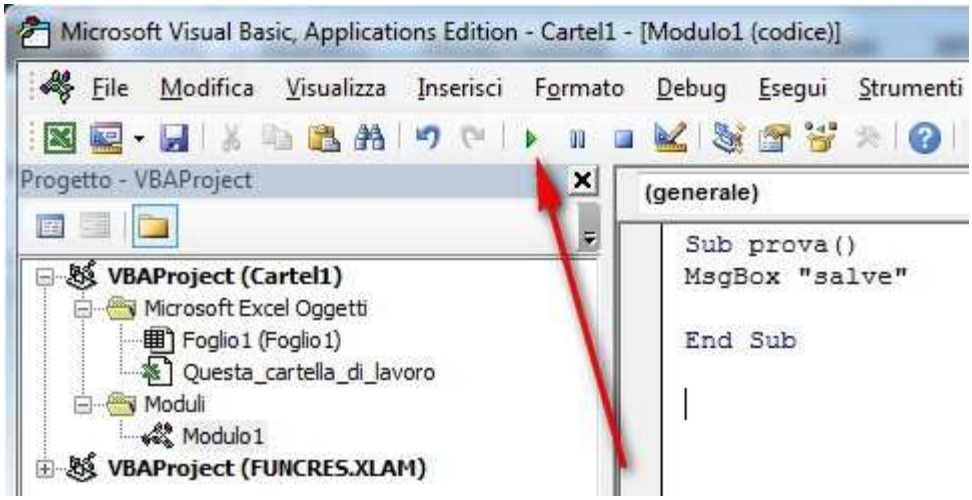


Fig. 6

A questo punto dobbiamo solo eseguire la macro che possiamo fare in questo modo:

- Cliccare in qualsiasi punto all'interno della macro
- Selezionare **Esegui-> Esegui Sub/UserForm**

Oppure**,** sempre posizionando il cursore in un punto qualsiasi della macro**,** cliccare sull’icona verde segnalata dalla freccia rossa di Fig. 6,

Nota: Se il cursore non si trova all'interno di una macro, allora verrà visualizzata una finestra con l’elenco delle macro disponibile viene chiesto di selezionarne una da mandare in esecuzione.

Mandando in esecuzione la macro si ottiene un risultato del genere:

Fig. 7

Le Variabili: Nozioni di Base

In programmazione, una variabile è un valore che viene affidato al computer per memorizzarlo temporaneamente nella sua memoria mentre il programma è in esecuzione. Si deve tener presente che la memoria del computer è suddivisa in piccole aree di stoccaggio utilizzate per contenere i valori delle applicazioni e quando si utilizza un valore nel codice, il computer lo memorizza in una di queste aree per poi rilasciarlo quando viene richiamato.

Immaginiamo che una variabile sia come una casella in cui si può inserire un dato di qualsiasi tipo e salvarlo per impiegarlo successivamente. Il nome della variabile è l'etichetta che identifica la casella e il contenuto della casella è il valore della variabile, la particolarità di una variabile è di poter cambiare il suo valore durante l'esecuzione della macro, mentre il nome rimane inalterato. Possiamo quindi sintetizzare che una variabile è il nome assegnato ad una specifica locazione di memoria del computer, e possiamo usare il nome della variabile per riferirci a qualsiasi dato contenuto in quella determinata locazione di memoria.

Il nome di una variabile deve essere scelto seguendo poche regole

- Deve cominciare con una lettera dell'alfabeto
- Dopo la prima lettera può contenere qualsiasi combinazione di numeri, lettere
- Il nome di una variabile non può contenere spazi, punti o caratteri speciali quali =, +, -, / e simili.
- Il nome della variabile non deve corrispondere a parole chiave di VBA
- Il nome di una variabile deve essere unico, cioè non può essere duplicato all'interno di un modulo

Per creare una variabile, è abbastanza semplice, da quanto abbiamo esposto finora basta solo dargli un nome ed assegnarli un valore, vediamone un esempio.

alex = 10

Questo enunciato memorizza il *valore 10* nella locazione di memoria denominata *alex*, se si tratta del primo enunciato VBA crea la variabile, riserva una locazione di memoria per contenere il dato della variabile e poi memorizza il valore 10 in questa nuova locazione di memoria specificato dal nome della variabile. Molto semplicemente basti ricordare che tutto ciò che si mette sul lato destro di un segno di uguale è quello che si sta tentando di memorizzare e tutto quello che si ha a sinistra del segno di uguale è il luogo in cui si sta cercando di conservarlo.
Se la variabile *alex* esiste già VBA memorizza il nuovo valore nella locazione di memoria a cui fa riferimento la variabile *alex*, in pratica sovrascrive il valore

Questa procedura è definita una *"dichiarazione implicita"*, oppure *"dichiarazione al volo"*, risulta molto comoda ma può presentare degli inconvenienti, infatti usando il metodo implicito la variabile creata da VBA è di tipo *Variant* (tutti i tipi di dati), inoltre se in seguito digitiamo il nome sbagliato (es. *Alex*), a seconda del punto in cui il nome sbagliato compare nel codice il VBA può generare un *errore di runtime*, oppure possiamo anche usare la sintassi corretta, ma così andremmo a distruggere il valore memorizzato precedentemente.

Allora come possiamo ovviare a questi inconvenienti? Dichiarando le variabili.

Dichiarazione di una variabile

E' indubbio che dichiarando le variabili ne ricaviamo notevoli benefici, ma l'errore umano nella digitazione del codice è sempre in agguato, per tutelarsi ulteriormente possiamo inserire un'altra parola chiave **Option Explicit**, se aggiungiamo questa parola chiave nell'area delle dichiarazioni di un modulo, cioè all'inizio del modulo prima di qualsiasi altra dichiarazione o listato, il VBA ci richiede di dichiarare tutte

le variabili tramite l'enunciato *Dim* prima di usarle, in pratica l'enunciato *Option Explicit* proibisce di dichiarare implicitamente variabili in ogni punto del modulo, possiamo dire che con l'istruzione Option Explicit abbiamo aggiunto un altro pezzettino al nostro listato per garantirne una perfetta esecuzione.

Una dichiarazione implicita contiene dati di tipo *Variant*, però il nostro obbiettivo è quello di abbinare le potenzialità di VBA per utilizzare o manipolare vari tipi di dati presenti nel nostro foglio di Excel, se per esempio volessimo eseguire una somma tra i dati contenuti in due variabili avremmo sicuramente una incompatibilità nei dati e quasi certamente ci verrà rimandato un errore. Per ovviare a questo ultimo inconveniente usiamo un'altra parola chiave nella dichiarazione della variabile e aggiungiamo anche il tipo di dati che andrà a contenere. La parola chiave è **As** e l'enunciato si presenta in questo modo:

Dim nome_variabile As tipo che nel nostro esempio diventa così *Dim alex As Integer*

Così facendo abbiamo creato una variabile di nome *alex* e abbiamo dichiarato che è di tipo numerico.
In una normale applicazione, non è raro dover utilizzare molte variabili e si dovrebbe prendere l'abitudine di dichiarare sempre una variabile prima di utilizzarla. Per dichiarare una nuova variabile dopo averne dichiarato una prima, si può semplicemente andare alla riga successiva e utilizzare la parola chiave Dim per dichiarare quella nuova. Ecco un esempio:

```
Dim pippo
Dim pluto
```

Allo stesso modo, è possibile dichiarare quante variabili vogliamo, inoltre è possibile dichiarare più variabili sulla stessa riga e per effettuare questa operazione, si utilizza sempre la parola chiave *Dim* separando i nomi delle variabili con una virgola. Ecco un esempio

```
Dim pippo, pluto, minni, alex
```

Vediamo un esempio di codice sul loro utilizzo

```
Sub var1()
Dim alex As String
alex = "Ciao a tutti"
MsgBox alex
End Sub
```

e otteniamo un messaggio del genere

Fig. 1

In questo modo abbiamo dichiarato la variabile all'interno della routine o sub *var1* e può essere usata solo in quella routine, infatti se usiamo questo codice

```
Sub var1()
Dim alex As String
alex = "Ciao a tutti"
stampa_box
End Sub

Private Sub stampa_box()
MsgBox alex
End Sub
```

otteniamo un messaggio del genere

Fig. 2

Non ci viene rimandato un errore, in quanto il comando MsgBox viene eseguito, ma non vediamo nessuna scritta, cioè la variabile alex non viene riconosciuta e non appare nel nostro box. Abbiamo parlato poco sopra di dichiarazione delle variabili nell'area di dichiarazione del modulo, assieme alla parola chiave *Option Explicit*, abbiamo anche già visto i benefici di questa particolare procedura, ma se in quell'area aggiungessimo anche la dichiarazione della variabile cosa succederebbe? Semplicemente che la variabile sarebbe condivisa e utilizzabile da tutte le routine di quel modulo. Vediamo un esempio modificando il codice del listato sopra esposto

```
Option Explicit
Dim alex As String
Sub var1()
alex = "Ciao a tutti"
stampa_box
End Sub

Private Sub stampa_box()
MsgBox alex
End Sub
```

Il listato va inserito nell'editor all'inizio del modulo come mostrato in *figura 3*

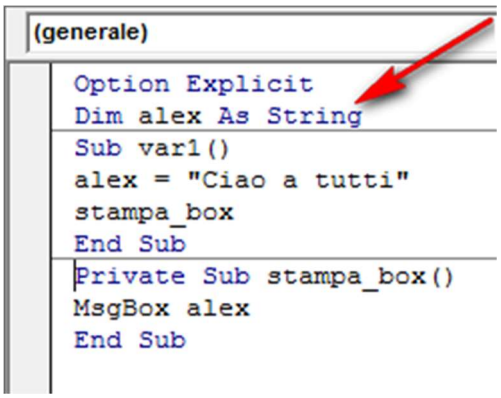


Fig. 3

Se eseguiamo questa macro verrà mostrato il contenuto della variabile in una finestra

Fig. 4

Avrete notato che nella routine principale è stato inserito il nome di un'altra routine cioè *stampa_box*, in questa forma la macro riconosce che quella è una chiamata ad un'altra macro e la esegue, inoltre la routine stampa_box è preceduta dalla funzione **Private**, che viene usata quando vogliamo utilizzare una routine di quel modulo e solo in quello, al tempo stesso questa ruotine non ci compare nella finestra di assegnazione delle macro.

Abbiamo visto l'utilità nell'inserire la parola chiave *Option Explicit* nel listato, è possibile evitare di inserire in ogni modulo tale riga di codice agendo nelle opzioni dell'editor per garantire che Option Explicit sia sempre inserito nella parte superiore del modulo operando in questo modo: dal menu **Strumenti - Opzioni** e nella finestra che ci viene mostrata mettere il flag alla voce *Dichiarazione di variabili obbligatoria*. Come mostrato in figura 5

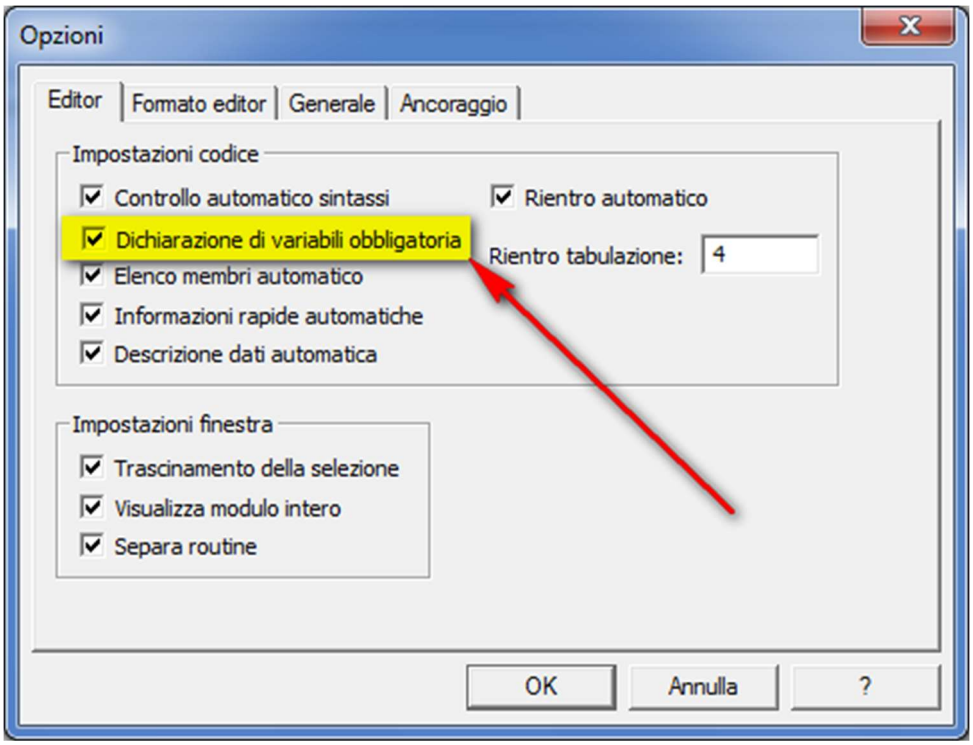


Fig. 5

Messa la spunta cliccare sul tasto Ok per confermare. È ora necessario utilizzare sempre la parola chiave Dim per dichiarare una variabile, in caso contrario verrà rimandato un errore di *"Variabile non definita"*. Abbiamo anche visto l'uso della funzione Private, quando facciamo programmi con listati lunghi e usiamo diversi moduli, ne facilita l'interpretazione del codice e il debug in caso di errore.

Abbiamo visto che tutte le variabili dichiarate all'interno di una procedura sono disponibili solo all'interno della procedura in cui le dichiarate, mentre quelle che dichiarate a livello di modulo (come in Fig. 3) sono disponibili a tutte le procedure all'interno del modulo in cui sono state dichiarate, ma *non sono disponibili* a procedure in un modulo diverso. In VBA gli elementi che sono disponibili a tutti i moduli vengono definiti a validità pubblica e sono chiamati variabili globali, perché sono disponibili globalmente cioè in tutto il vostro programma tramite la parole chiave **Public** usando la seguente sintassi

Public NomeVariabile [As NomeTipo]

Dove *NomeVariabile* rappresenta un nome valido qualsiasi per identificare la variabile e *NomeTipo* un qualsiasi nome di tipo di dato valido per fare un esempio vediamo il listato di prima modificandone la dicitura con la parola chiave *Public* come mostrato in figura 6

```
(generale)
Option Explicit
Public alex As String
Sub var1()
alex = "Ciao a tutti"
stampa_box
End Sub
Private Sub stampa_box()
MsgBox alex
End Sub
```

Fig. 6

Utilizzare Public per dichiarare una variabile a livello globale può essere utile quando si ramifica il programma in diversi moduli ma è da usare con molta attenzione specialmente nell'assegnazione del nome alla variabile per evitare di creare confusione se esistono due variabili con lo stesso nome in moduli diversi, pertanto cercate di essere espliciti nell'uso della variabile nella forma Public

Tuttavia, si deve tener presente che ci troviamo in ambiente Excel, un foglio di calcolo, con celle che contengono valori, per cui dobbiamo chiederci anche cosa si vuole fare di questa variabile e del valore in esso contenuto. Dopo tutto, che senso avrebbe memorizzare un valore, se non lo si vuole usare? Quello che possiamo fare è di trasferire il nostro valore memorizzato nella variabile in una cella del foglio di calcolo.
Precedentemente abbiamo visto la proprietà di Range che tramite la proprietà Value può ottenere o impostare il valore di una cella, o un gruppo di cellule.
Per impostare un valore, la proprietà Range va messa prima di un segno di uguale, in questo modo:

Sheets("Foglio1") .Range ("A1"). Value =

In questo modo vogliamo puntare al Range A1 del foglio con nome "Foglio1" e dopo il punto, dobbiamo quindi inserire il valore da assegnare alla cella A1 tramite la proprietà Value e dopo il segno di uguale, è possibile digitarne il valore in questo modo:

Sheets("Foglio1") .Range ("A1"). Value = 10

Ora, il valore per la cella A1 sul foglio di lavoro "Foglio1" è impostato su 10
Per assegnare un valore è possibile digitare il nome di una variabile invece di digitare un numero. Se nel listato del codice abbiamo precedentemente assegnato il valore 10 alla variabile "var1" possiamo esprimere il listato in questo modo:

Sheets("Foglio1") .Range ("A1"). Value = var1

VBA vede la variabile denominata var1 quindi recupera qualsiasi valore memorizzato all'interno di esso e tale valore viene quindi memorizzato all'interno di ciò che è a sinistra del segno uguale.

Variabili e tipi di dati

Prima di addentrarci nell'argomento della lezione faremo una panoramica su come vengono memorizzati vari tipi di informazioni come numeri, date e testi, in modo che possano essere distinti tra di loro e come possano essere manipolati ed elaborati. Per chiarire meglio l'argomento e cercare di capire di cosa stiamo parlando è opportuno occuparsi ora delle diverse tipologie di dati che può trattare VBA e anche dello spazio che questi dati occupano in memoria. Faremo ora alcune considerazioni sulla distinzione dei tipi di dato (tipo di un dato è il termine che fa riferimento alla particolare natura dei dati che il VBA può memorizzare e manipolare quali testo e numeri) e cercheremo di vedere come può un programma lavorare con dati di tipo diverso come stringhe (la stringa è una sequenza di caratteri di testo) e numeri.

Possiamo comprendere meglio quanto esposto e la sua importanza con un esempio. Prendiamo una sequenza di caratteri *"17081974"* a prima vista sembra trattarsi di un numero, ma potrebbe anche rappresentare una data, il 17 Agosto del 1974 oppure potrebbe essere un numero di telefono (170 81 974). Come facciamo allora a determinare che cosa rappresenta quella stringa?
Possiamo determinare cosa rappresenta la stringa in base all'uso che ne dobbiamo fare del dato. Se si tratta di un importo sicuramente usato per eseguire dei calcoli, di conseguenza si tratta di un numero, se invece rappresenta una data verrà utilizzata in modo particolare, si può sommare ma va trattata in modo diverso oppure se è un numero di telefono sappiamo che non può essere trattato per il calcolo anche se è costituito da una sequenza numerica

Da queste considerazioni nasce l'esigenza o la necessità di dover sapere sempre la natura e lo scopo di un dato all'interno di un programma. I vari tipi di dati possono essere di tipo: **Booleano, Byte, Date, String, Integer, Single, Long, Double, Currency, Variant**, I più utilizzati sono:

- Byte : E' di tipo numerico intero senza segno compreso nell'intervallo da 0 a 255
- Integer : E'usato per rappresentare numeri interi (con segno) compresi fra -32768 a 32767
- String : Può contenere delle sequenze di caratteri (stringhe) a lunghezza variabile oppure a lunghezza fissa
- Long : Rappresenta numeri interi compresi fra -2,147,483,648 e 2,147,483,647,
- Single e Double : Utilizzato per memorizzare numeri reali a singola o a doppia precisione
- Boolean : E' di tipo logico che può assumere il valore TRUE o FALSE (vero o falso)
- Date : E' utilizzato per memorizzare data e ora
- Variant : E' un tipo universale che può contenere dei dati di qualsiasi formato

Questi vari tipi di dati vengono memorizzati da VBA per poterli utilizzare come sotto forma di *variabile* che consentono di memorizzare temporaneamente dei valori durante l'esecuzione di un'applicazione. Alle variabili deve essere associato un nome, utilizzato per fare riferimento al valore della variabile, e un tipo di dati che determina la modalità di memorizzazione dei bit che rappresentano i valori nella memoria del computer.

Tipo di dati

Un tipo di dati indica al computer che tipo di variabile si intende utilizzare perché prima di utilizzare una variabile, si dovrebbe sapere quanto spazio occuperà in memoria. Diverse variabili utilizzano diverse quantità di spazio in memoria e le informazioni che specificano la quantità di spazio di cui ha bisogno la variabile è chiamato tipo di dati e viene misurato in byte. Per specificare il tipo di dati che verrà utilizzato per una variabile, dopo aver digitato la parola chiave Dim seguito dal nome della variabile si deve digitare la parola chiave As seguita da uno dei tipi di dati che esamineremo successivamente. La formula utilizzata è: *Dim nome_variabile As tipo_dati*

Abbiamo detto in precedenza che è possibile utilizzare diverse variabili se sono necessarie e quando si dichiara tali variabili, si possono dichiarare su righe separate e per specificare il tipo di dati si utilizza la stessa formula di cui sopra. Abbiamo anche visto che si possono dichiarare varie variabili sulla stessa riga a patto che siano separate con una virgola, ora se si specifica il tipo di dati di ogni variabile si usa la stessa regola, si digita la virgola dopo ogni variabile in questo modo:
Dim Nome As tipo_dati, Cognome As tipo_dati

Il codice sopra esposto appare come se vi fosse un solo tipo di dati, di seguito passeremo in rassegna i vari tipi di valori che sono disponibili, e per dichiarare le variabili di tipi di dati diversi, si dichiara ognuna su una riga come abbiamo visto in precedenza. È inoltre possibile dichiarare variabili di tipi di dati diversi sulla stessa linea e per fare questo, si utilizza sempre la parola chiave Dim separando le dichiarazioni con le virgole.

Tipo di Caratteri

Per rendere la dichiarazione della variabile più veloce e anche conveniente, è possibile sostituire l'espressione *As tipo_dati* con un carattere speciale che rappresenta il tipo di dati previsto. Questo carattere si chiama Tipo di carattere e dipende dal tipo di dati che si intende applicare a una variabile e se viene utilizzato, il tipo di carattere deve essere l'ultimo carattere del nome della variabile. Vedremo quali caratteri sono disponibili e quando possono essere applicati

Valore di conversione

Ogni volta che l'utente inserisce un valore in un'applicazione, tale valore viene considerato in primo luogo come testo, ciò significa che, se si desidera utilizzare tale valore in un'espressione o un calcolo che prevede un valore specifico diverso dal testo, è necessario convertire quel testo. Fortunatamente, Microsoft Visual Basic fornisce un meccanismo efficace per convertire un valore di testo di uno degli altri valori che vedremo dopo. Per convertire il testo ad un altro valore, vi è una parola chiave adatta allo scopo e che dipende dal tipo di valore in cui si desidera convertirlo.

Variabili numeriche - Integer

Se vogliamo utilizzare un numero nel programma, Visual Basic è in grado di riconoscere un numero naturale qualsiasi che non include una parte frazionaria e il numero è costituito da una combinazione di 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9, nessun altro carattere è consentito.

Byte

Per dichiarare una variabile che conterrebbe numeri naturali che vanno da 0 a 255, si utilizza come tipo di dati Byte. In questo modo:*Dim AnniServizio As Byte*
Non esiste un tipo di carattere per il tipo di dati Byte e dopo aver dichiarato la variabile, è possibile assegnare un piccolo numero positivo.

```
Dim Valore As Byte
Valore = 246
```

È inoltre possibile utilizzare il numero in formato esadecimale fino a quando il numero è inferiore a 255 e se si dà un valore negativo o un valore superiore a 255, quando si tenta di accedervi, si riceverà un errore. Per convertire un testo in un numero piccolo, è possibile utilizzare la funzione **CByte ()** utilizzando la seguente formula: *Numero = CByte (Valore da convertire a Byte)* si ricorda che quando si utilizza CByte (), si deve immettere il valore da convertire tra parentesi.

Integer

Per dichiarare una variabile che potrebbe contenere un numero che varia -32.768 a 32.767, si utilizza come tipo di dati Integer con questa sintassi: *Dim conta As Integer*
Invece di utilizzare l'espressione As Integer, è possibile utilizzare il carattere **%** come tipo di dati, pertanto, la dichiarazione di cui sopra può essere fatta come segue: *Dim conta %*

Dopo aver dichiarato la variabile, è possibile assegnarle il valore desiderato, se si assegna un valore inferiore a -32768 o superiore a 32767, quando si decide di usarla, si riceverà un errore. Se si vuole convertire un testo in un numero, si può usare la funzione **CInt ()** utilizzando la seguente formula: *Numero = CInt (Valore da convertire)* dove tra le parentesi di CInt () si deve inserire il testo, o l'espressione che deve essere convertita.

Long

Un intero Long è un numero che può essere utilizzato per una variabile che coinvolge un numero maggiore di Integer, pertanto per dichiarare una variabile che potrebbe contenere un numero così elevato, si utilizza il tipo di dati Long in questo modo: *Dim Popolazione As Long*
Il tipo di carattere per il tipo di dati Long è **@**, pertanto la variabile di cui sopra può essere dichiarata come segue: *Dim Popolazione @*

Una variabile Long può memorizzare un valore compreso tra -2,147,483,648 e 2,147,483,647 (le virgole sono usate per facilitare la lettura, non devono essere utilizzati nel codice), pertanto, dopo aver dichiarato una variabile Long, è possibile assegnare un numero in tale intervallo. Per convertire un valore testo in un intero Long, si usa l'espressione **CLng ()** utilizzando la seguente formula: *Numero = CLng (Valore da convertire)*, ricordando di inserire nelle parentesi di CLng () il testo da convertire

Single

Nella programmazione, un numero decimale è quello che rappresenta una frazione, esempi sono 1.85 e 426,88. Se si prevede di utilizzare una variabile che sarebbe di quel tipo di numero, ma la precisione non è la vostra preoccupazione principale, si dichiara come tipo di dati Single così: *Dim Distanza As Single*
Il tipo di carattere per il tipo di dati Single è **!** pertanto la dichiarazione di cui sopra potrebbe essere scritta in questo modo: *Dim Distanza !*

Se si dispone di un valore testo che deve essere convertito, si utilizza la funzione **CSng ()** con la seguente formula: *Numero = CSng (Valore da convertire)*, nelle parentesi di CSng () immettere il valore da convertire.

Double

Se si desidera utilizzare un numero decimale che richiede una buona dose di precisione, si dichiara una variabile con tipo di dati Double con questa sintassi: *Dim Distanza As Double* . Invece di As Double, è possibile utilizzare il tipo di carattere **#** in questo modo: *Dim Distanza #*

Per convertire un valore testo in Double si utilizza la funzione **CDbl ()** con la seguente formula: *Numero = CDbl (Valore da convertire)* inserendo nelle parentesi di CDbl () , il valore che deve essere convertito..

String

Una stringa è un carattere o una combinazione di caratteri che costituiscono il testo di qualsiasi tipo e quasi qualsiasi lunghezza. Per dichiarare una variabile di tipo stringa, si utilizza come tipo di dati String in questo modo: *Dim Paese As String*. Il tipo di carattere per i dati String è **$**, pertanto, la dichiarazione di cui sopra può essere scritta come: *Dim Paese $*

Come già detto, dopo aver dichiarato una variabile, è possibile assegnarle un valore, nel caso di una variabile stringa il valore deve essere incluso all'interno di doppi apici in questo modo:

```
Dim Paese As String
Paese = "Italia"
```

Se si dispone di un valore che non è in formato testo e si desidera convertirlo in una stringa, si deve utilizzare la funzione **CStr ()** con la seguente formula: *CStr (valore da convertire in stringa)*, inserendo nelle parentesi di CStr (), il valore che si desidera convertire in stringa.

Currency - Valuta

Il tipo di dati valuta viene utilizzato per trattare valori monetari. Ecco un esempio di dichiarazione: *Dim Salario As Currency*. Invece di utilizzare la valuta come espressione, è possibile utilizzare il carattere **@** come tipo di dati per dichiarare una variabile di valuta così: *Dim Salario @*
Quando si assegna un valore a una variabile Currency non si deve utilizzare il simbolo di valuta. Ecco un esempio di assegnazione di un numero di valuta per una variabile:

Quando si assegna un valore a una variabile **Currency** non si deve utilizzare il simbolo di valuta. Ecco un esempio di assegnazione di un numero di valuta per una variabile:

```
Dim Salario As Currency
   Salario = 66500
```

Se si desidera convertire un valore in Currency, si deve utilizzare la funzione **CCur ()** con la seguente formula: *Number = CCur (Valore da convertire)*, inserendo il valore da convertire tra le parentesi di CCur ().

Date - Data

Una data come tipo di dati può essere utilizzata per memorizzare un valore data, pertanto, per dichiarare una variabile data si utilizza come tipo di dati Date in questo modo: *Dim Data_nascita As Date*. Dopo aver dichiarato la variabile, è possibile assegnare un valore, che deve essere compreso tra due simboli **#**(cancelletto). Ecco un esempio:

```
      Dim Data_nascita As Date
   Data_nascita = # 10/8/1988 #
```

Se si dispone di una stringa o un'espressione che si desidera convertire in un valore data, si deve utilizzare la funzione **CDate ()** in base alla seguente formula: *Risultato = CDate (Valore da convertire)*, inserendo nelle parentesi di CDate (), il valore che deve essere convertito.

Time

In Visual Basic, il tipo di dati Date può essere utilizzato anche per memorizzare un valore di tempo, ecco un esempio di dichiarazione di una variabile che può contenere un valore di tempo: *Dim tempo As Date*
Per assegnare un valore alla variabile si segue la sintassi della funzione Date

Variant

Fino ad ora, abbiamo dichiarato variabili conoscendone il tipo di valore che dovevano contenere, VBA offre un tipo di dati universale che è possibile utilizzare per qualsiasi tipo di valore, il tipo di dati Variant. Questo tipo di dati viene utilizzato per dichiarare una variabile il cui contenuto non è esplicitamente specificato, ciò significa che un tipo di dati Variant può contenere qualsiasi tipo di valore che si desidera. Ecco alcuni esempi di variabili di tipo Variant, che contengono diversi tipi di valori:

```
Sub Test ()
   Dim Nome, Paese, Salario, DataN As Variant
   Nome = "Eva Kant"
   Paese = 2
   Salario = 35.65
   DataN = # 24/02/2004#
End Sub
```

Durata di una variabile

Fino ad ora, abbiamo dichiarato le variabili tra le linee di codice Sub e End Sub, questo tipo di variabile è definita come variabile locale. Una variabile locale è limitata alla zona in cui è dichiarata, in pratica non è possibile utilizzare tali variabili al di fuori della Sub in cui è stata dichiarata. Ecco un esempio:

```
Option Explicit
Sub Test ()
   Dim Nome As String
   Nome = "Eva"
End Sub
```

Variabili globali

Una variabile globale è una variabile dichiarata al di fuori della Sub e questo tipo di variabile è normalmente dichiarata nella sezione superiore del file. Ecco un esempio:

```
Option Explicit
Dim Cognome As String

Sub Test ()
  .....
End Sub
```

Dopo aver dichiarato una variabile globale, è possibile accedervi dalle altre aree del file. Ecco un esempio:

```
In Modulo 1
Option Explicit
Dim Cognome As String

In Modulo2

Sub Test ()
   Dim Nome As String
   Nome = "Eva"
   Cognome = "Kant"
End Sub
```

Anche se abbiamo dichiarato la nostra variabile globale all'interno del file in cui è stato utilizzato, è anche possibile dichiarare una variabile globale in un modulo separato per essere in grado di utilizzarlo in un altro modulo.

Variabili private

Una variabile è indicata come Private se può accedere solo al codice all'interno dello stesso modulo, dove viene utilizzato. Per dichiarare una tale variabile, invece della parola chiave Dim, si utilizza la parola chiave **Private**. Ecco un esempio:

```
Option Explicit
Private Cognome As String

Sub Test ()
    Dim Nome As String
    Nome = "Eva"
    Cognome = "Kant"
End Sub
```

Ricordate che una variabile privata è accessibile da qualsiasi codice nello stesso modulo

Variabili pubbliche

Una variabile è denominata **Public** se si può accedere al codice sia all'interno dello stesso modulo in cui è dichiarata o dal codice esterno relativo modulo. Per dichiarare una variabile pubblica, invece della parola chiave Dim, si utilizza la parola chiave **Public**. Ecco un esempio:

```
Option Explicit
Private Cognome As String
Public Nome_esteso As String

Sub Test ()
    Dim Nome As String
    Nome = "Eva"
    Cognome = "Kant"
    Nome_esteso = Nome & "" & Cognome
End Sub
```

Come promemoria possiamo affermare che una variabile pubblica è disponibile per codificare dentro e fuori dal suo modulo, ciò significa che è possibile creare un modulo, dichiarare una variabile pubblica, e accedere a quella variabile da un altro modulo, mentre una variabile privata è disponibile all'interno del suo modulo, ma non al di fuori dello stesso. Se si dichiara una variabile privata in un modulo e si prova ad accedervi da un altro modulo, si riceverà un errore:

```
In Modulo 1
Option Explicit
Private Cognome As String

In Modulo 2
Option Explicit
Private Nome As String

Sub Test ()
    Nome = "Eva"
    Cognome = "Kant"
    Nome_esteso = Nome & "" & Cognome
    ActiveCell.FormulaR1C1 = Nome_esteso
End Sub
```

Gestione degli errori - Metodi e proprietà

La *gestione degli errori* si riferisce ad una particolare tecnica di programmazione che consiste nell'anticipare e prevedere le condizioni di errore che possono sorgere quando il programma viene eseguito. In generale gli errori possono essere di tre tipi:

- Errori di sintassi: Come errori di battitura o variabili non dichiarate che impediscono il corretto funzionamento
- Errori run-time: Che si verificano quando VBA non può eseguire correttamente una dichiarazione del programma.
- Errori Logici: Come un utente immette un valore negativo in cui solo un numero positivo è accettabile

Errori di sintassi

Viene detta sintassi lo specifico ordine di parole che costituisco un enunciato VBA valido e alcuni dei più comuni messaggi di errore che possono verificarsi mentre si scrive o si edita una procedura sono relativi a errori di sintassi. I messaggi di errore di sintassi segnalano un problema in un enunciato quali, virgole, virgolette argomenti mancanti o simili. Ogni volta che si scrive una nuova riga di codice o se ne modifica una, il VBA esamina la riga non appena il cursore si sposta da essa. Questa operazione che viene detta ***Parsing*** è il processo di scomposizione di un enunciato nelle varie parti al fine di determinare quali sono le parole chiave, le variabili o i dati. Dopo che il VBA ha esaminato senza rilevare errori una riga di codice, procede alla sua compilazione (in VBA la compilazione è la conversione del codice sorgente in una forma direttamente eseguibile dal VBA senza dover ripetere l'operazione di Parsing)

Si deve tenere presente che la finestra del codice nell'Editor di VB presenta le parole scritte in vari colori, questa "applicazione" del colore viene eseguita dopo aver esaminato una riga di codice e non aver rilevato errori, al contrario, se viene rilevato un errore in fase di esame o compilazione della riga il VBA colora di rosso l'intera riga e visualizza una finestra di dialogo con un messaggio di errore

Errori Run-time

E' possibile scrivere un enunciato VBA senza alcun errore di sintassi, ma che tuttavia non porta a un'esecuzione corretta, gli errori che si manifestano solo in fase di esecuzione della procedura vengono detti errori runtime. Ne esistono di vari tipi e di solito sono causati dalla mancanza di argomenti o dall'uso di argomenti di tipo errato, dalla mancanza di certe parole chiave o dal tentativo di accedere a dischi o directory non esistenti o da errori logici.

Errori logici

Questo tipo di errore è il più difficile da tracciare se la sua sintassi è corretta e funziona senza rimandare errori di esecuzione, si può definire come un errore da nessuna indicazione all'utente che si è verificato un errore dovuto al fatto che un errore logico è il processo di logica e non il codice stesso a generare l'errore. L'esecuzione di un calcolo in un foglio di lavoro utilizzando una funzione restituirà una risposta, ma è la risposta corretta? Per meglio comprendere simuliamo un errore scrivendo il codice sotto riportato in un modulo e mandiamolo in esecuzione

```
Sub prova()
Sheets("Foglio11").Range("F5").Select
End Sub
```

Il cui significato è: *vai nel Foglio 11* di questa cartella e seleziona (*Select*) la cella F5 *Range("F5")*. Tornando ad Excel (basta cliccare sulla prima icona col simbolo di Excel in alto a sinistra) e mandando in esecuzione la macro (premere **ALT+F8**, selezionare il nome della macro e poi *Esegui*) e comparirà un avviso come quello sotto riportato

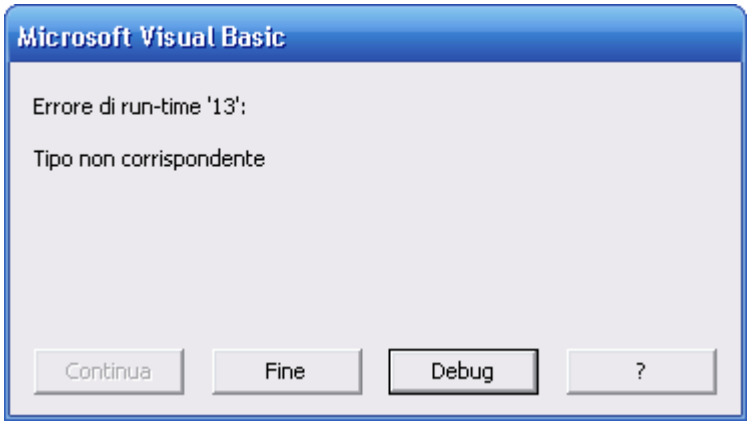


Fig 1

Come si può intuire dal testo si tratta di un errore generato dal codice, premendo sul pulsante **Debug** verrà automaticamente aperto l'editor di VBA e verrà riportata la routine (o Sub) che lo ha causato

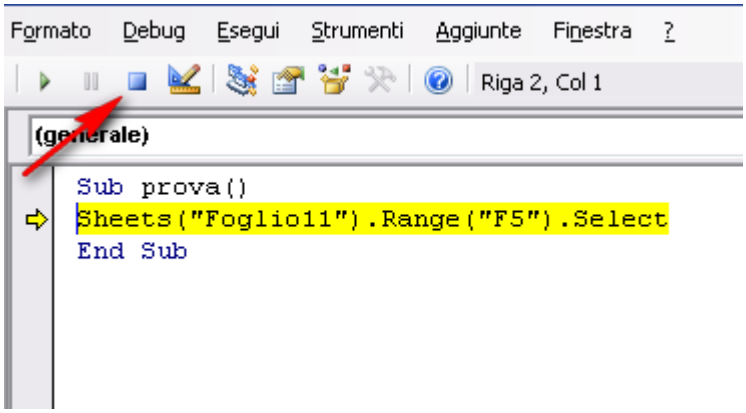


Fig. 2

Nella figura sopra riportata si può notare che la riga di codice che ha causato l'errore è già evidenziata in giallo e questo facilita notevolmente le cose, l'errore è già individuato e si può intervenire e correggere il codice, tuttavia l'errore ha bloccato l'esecuzione della macro e per poter continuare ad operare si deve ripristinare l'editor (cioè sbloccarlo) e in seguito rimediare all'errore. Per eseguire questa operazione basta premere sul pulsante blu in alto nella barra degli strumenti contrassegnato dalla freccia rossa, il quale interrompe il Debug del codice e ripristina l'uso dell'editor.

Spieghiamo ora la natura dell'errore: nella nostra cartella di lavoro, l'errore è stato causato per il semplice fatto che **non esiste** all'interno della stessa un foglio denominato *Foglio11*, per cui VBA ha evidenziato questa situazione con un errore di run-time interrompendo l'esecuzione della macro, in pratica non ha trovato il percorso che gli è stato indicato. Esaminiamo ora il seguente codice:

```
Sub prova()
MsgBox "Ciao Mondo", "Messaggio di saluto"
End Sub
```

Mandando in esecuzione la routine viene rimandato lo stesso avviso del codice precedente

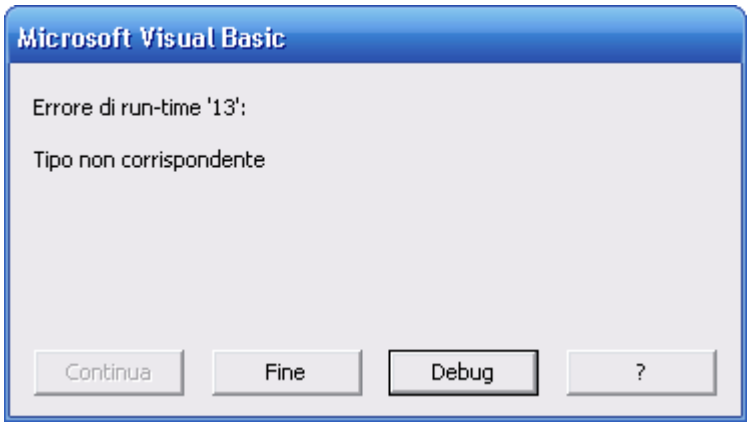


Fig. 3

In questo caso non ci sono errori di sintassi nell'enunciato *MsgBox*, i testi sono correttamente posti tra virgolette, ed è presente la virgola di separazione tra gli argomenti (Ndr. Tutti gli argomenti di MsgBox, salvo il primo, sono opzionali pertanto VBA accetta la presenza di due soli argomenti per MsgBox come sintassi corretta), ma quando VBA cerca di eseguire l'enunciato visualizza una finestra di dialogo di errore come sopra esposta.

Questa finestra di dialogo informa che l'errore si è verificato in fase di esecuzione della procedura e contiene un messaggio che descrive l'errore stesso, in questo caso si tratta di un argomento di tipo errato (non corrispondente). In effetti esaminando la riga, si vede che manca lo spazio che funge da segnaposto per il secondo argomento, *Buttons*, facoltativo. Quando VBA sottopone al Parsing l'enunciato, compila il testo fra virgolette "Messaggio di saluto" come secondo argomento di MsgBox invece di terzo argomento, a causa dell'omissione del segnaposto fra le due virgole. Dato che l'argomento buttons deve essere un numero e non un testo, il VBA segnala che il tipo di dati passato alla procedura MsgBox non è del tipo giusto per quell'argomento.

E' possibile intercettare un errore nel progetto VBA con l'istruzione **On Error** per deviare il flusso del programma verso una routine che gestisca l'errore trasmettendo le istruzioni necessarie per risolverlo. Per poter controllare un errore si inserisce all'inizio della routine l'istruzione *On Error GoTo [etichetta]* e quando si verifica un errore, questa parola chiave interrompe l'esecuzione del codice del programma per "saltare" alla posizione contrassegnata da [etichetta] che deve trovarsi nella stessa procedura che contiene l'istruzione *On Error*.

Per esempio se la routine Command1_Click ha come prima istruzione un *On error goto err* (dove err è il nome dell'etichetta) in caso di errore il flusso del programma va direttamente all'istruzione presente nell'etichetta err, dove si deve inserire il codice o la procedura che tratteranno opportunamente l'errore. Se invece tutto il flusso del programma procede correttamente, l'istruzione On Error viene ignorata e si arriva alla fine della routine. Ci sono vari modi per gestire un errore utilizzando le seguenti istruzioni:

- *On Error GoTo 0 :* Indica al programma di ignorare l'istruzione in caso di errore e il programma continuerà
- *On Error Resume :* Indica che in caso d'errore si ritenta di eseguire l'istruzione che lo ha generato
- *On Error Resume Next :* Indica al programma che in caso d'errore verrà eseguita l'istruzione successiva a quella che ha generato l'errore

E' abbastanza chiaro che solo utilizzando l'istruzione *On Error GoTo [etichetta]* si può trattare l'errore perché negli altri 3 enunciati la situazione si risolve a prescindere dal tipo di errore, in pratica si salta l'errore ma non lo si risolve.

Ad ogni modo è consigliabile porre gli enunciati di gestione degli errori alla fine di ogni procedura inserendo delle funzioni tipo *Exit Sub* oppure *Exit Function*. Vediamo un esempio di codice in un routine con gestione degli errori.

```
Sub Command1_Click()
On Error GoTo err
  MsgBox "Ciao Mondo", "Messaggio di saluto"
Exit_Command1:
Exit Sub

err:
MsgBox Err.Description
Resume Exit_Command1
End Sub
```

Nel codice di esempio sopra esposto *On Error* attiva la gestione degli errori, se si verifica un errore la procedura salta all'etichetta di riga *err* che indica l'inizio della gestione degli errori. La prima riga riporta una finestra di dialogo che visualizza il codice di errore, poi passa all'istruzione *Resume* che ci porta alla routine Exit_Command1 che con l'istruzione Exit Sub ci fa uscire dalla procedura. Se eseguiamo il codice sopra riportato ci riporta un avviso del genere.

Fig. 4

La forma *On Error Goto 0*, è la modalità predefinita in VBA e indica che quando si verifica un errore di runtime VBA dovrebbe visualizzare la sua finestra di messaggio di errore in fase di esecuzione standard, consentendo di immettere il codice in modalità di Debug o di terminare il programma VBA. In pratica avere On Error Goto 0 è come non avere un gestore degli errori attivato, in quanto qualsiasi errore causerà VBA verrà visualizzata la casella di messaggio di errore standard.

La forma*, On Error Resume Next*, è la forma più comunemente usata e abusata, con questa espressione si insegna a VBA di ignorare essenzialmente l'errore e riprendere l'esecuzione sulla riga successiva di codice. E 'molto importante ricordare che On Error Resume Next non risolve in nessun modo l'errore, ma indica semplicemente a VBA di continuare come se non ci fosse stato nessun errore. Tuttavia, l'errore può avere effetti collaterali, come variabili o oggetti impostati su Nothing non inizializzate ed è una responsabilità del codice di verificare una condizione di errore e prendere i provvedimenti opportuni. A tale scopo, vedendo il codice sotto riportato si deve testare il valore di *Err.Number* e se non è zero si deve eseguire la correzione appropriata. Per esempio:

```
On Error Resume Next
   N = 1/0
   If Err.Number <> 0 Then
      N = 1
   End If
```

Questo codice tenta di assegnare il valore 1/0 alla variabile N e questo è un'operazione non ammessa, e VBA genererà un errore 11 (divisione per zero) e dato che abbiamo l'espressione On Error Resume Next, il codice continua, ma in seguito viene verificato il valore di Err.Number e viene assegnato un altro valore alla variabile N

La forma *On Error Goto [etichetta]* indica a VBA di trasferire l'esecuzione alla riga che segue l'etichetta di riga specificata e ogni volta che si verifica un errore, l'esecuzione del codice va subito alla linea che segue l'etichetta di riga e non viene eseguita nessuna istruzione presente nel tra l'errore e l'etichetta, incluse le affermazioni di controllo del ciclo.

```
On Error Goto Err11:
N = 1/0
  'altro codice
Exit Sub
Err11:
 'codice di gestione degli errori
Resume Next
End Sub
```

Il gestore degli errori abilitato e attivo

Un gestore di errori è detto attivato quando un'istruzione On Error viene eseguita e indica il codice che viene eseguito quando si verifica un errore e l'esecuzione viene trasferita in un'altra posizione tramite la dichiarazione On Error Goto [etichetta]. Possiamo definire un blocco di gestione degli errori, chiamato anche gestore di errori, una sezione del codice che prende il controllo del programma attraverso una dichiarazione On Error Goto [etichetta]. Questo codice deve essere progettato sia per risolvere il problema e riprendere l'esecuzione nel blocco di codice principale o di interrompere l'esecuzione della procedura, non è possibile utilizzare On Error Goto [label] semplicemente per "saltare" tra le linee di codice. Ad esempio, il seguente codice non funzionerà correttamente:

```
On Error GoTo Err1:
Debug.Print 1/0
 'altro codice
Err1:
On Error GoTo Err2:
Debug.Print 1/0
 'altro codice
Err2:
```

Quando si verifica il primo errore, il flusso del programma viene trasferito alla linea Err1 ma l'errore è ancora attivo quando si verifica il secondo errore, e quindi quest'ultimo non viene intercettato dalla istruzione On Error.

L'istruzione Resume

La dichiarazione **Resume** indica al VBA un punto specifico del codice dove riprendere l'esecuzione, inoltre è possibile utilizzare Resume solo in un blocco di gestione degli errori, qualsiasi altro uso causerà un errore. Si deve considerare che *Resume* è l'unico modo, a parte uscire dalla procedura, per uscire da un blocco di gestione degli errori. Si deve ricordare di **non** utilizzare la dichiarazione *Goto* per dirigere l'esecuzione del codice da un blocco di errore, agire in questo modo si causeranno strani problemi ai gestori di errori. La dichiarazione Resume può assumere tre forme sintattiche

- Resume
- Resume Next
- Resume [Etichetta]

Usato da solo, Resume provoca l'esecuzione di riprendere alla riga di codice che ha causato l'errore, in questo caso è necessario assicurarsi che il blocco di gestione degli errori risolva il problema che ha causato l'errore iniziale, in caso contrario, il codice entrerà in un ciclo infinito, saltando tra la riga di codice che ha causato l'errore e il blocco di gestione degli errori. Il codice seguente tenta di attivare un foglio di lavoro che non esiste, questo provoca un errore, e il codice salta al blocco di gestione degli errori che crea il foglio, correggere il problema, e riprende l'esecuzione alla riga di codice che ha causato l'errore.

```
On Error GoTo Err1:
   Worksheets("Foglio11").Activate
   Exit Sub

   Err1:
   If Err.Number = 9 Then
      'il foglio non esiste, quindi lo crea
      Worksheets.Add.Name = "Foglio11"
     'torna alla riga di codice che ha causato il problema
      Resume
   End If
```

La seconda forma di Resume è **Resume Next**, che riprende l'esecuzione del codice nella riga immediatamente successiva alla riga che ha causato l'errore. Il seguente codice genera un errore (11 - divisione per zero) quando si tenta di impostare il valore di N. Il blocco di gestione degli errori assegna il valore 1 alla variabile N, e poi riprende l'esecuzione del codice con l'istruzione dopo l'istruzione che ha causato l'errore.

```
On Error GoTo Err1:
   N = 1 / 0
   Debug.Print N
   Exit Sub

   Err1:
   N = 1
   'torna alla riga successive che ha causato l'errore
   Resume Next
```

La terza forma di Resume è **Resume [etichetta]** che porta l'esecuzione del codice a riprendere in un'etichetta di riga, questo permette di saltare una sezione di codice se si verifica un errore.
Per esempio:

```
On Error GoTo Err1:
   N = 1 / 0
      'codice che viene saltato se si verifica un errore

   Label1:
   'codice da eseguire
   Exit Sub

   Err1:
   'ritorna alla linea Label1
   Resume Label1:
```

Ogni procedura non deve necessariamente avere un codice di gestione degli errori quando si verifica un errore, VBA utilizza l'ultima istruzione On Error per dirigere l'esecuzione del codice, tuttavia, se la procedura in cui si verifica l'errore non ha un gestore degli errori, VBA guarda indietro attraverso le chiamate di procedura che portano al codice errato. Ad esempio, se la procedura A chiama B e B chiama C, e A è l'unica procedura con un gestore degli errori, se si verifica un errore nella procedura C, l'esecuzione di codice viene immediatamente trasferito al gestore di errore nella procedura A, saltando il codice rimanente in B

I Menù dell'Editor di VBA

Vediamo ora una panoramica dei comandi dei menù e della barra strumenti dell'editor di VB per fare una carrellata delle capacità dell'editor stesso. Nella linea sotto l'intestazione appare la barra con i menu: **File, Modifica, Visualizza, Inserisci, Formato, Debug, Esegui, Strumenti, Aggiunte, Finestra?**

Menù File

Il menu File come in tutte le applicazioni Windows, contiene i comandi relativi al salvataggio e all'apertura dei file. Nell'Editor VB, il menu file offre i comandi necessari per salvare le modifiche apportate ai progetti VBA e stampare il codice sorgente delle macro

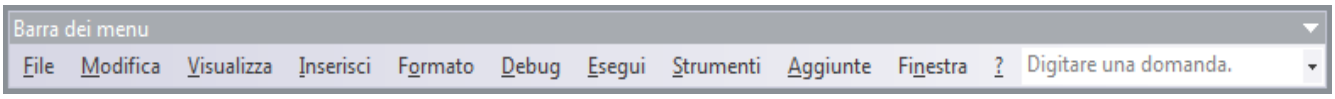


Fig. 1

Ogni menu è di tipo a tendina e si apre con un clic del mouse, mostrando una serie di comandi. I comandi possono essere momentaneamente disattivati e appaiono sbiaditi (scritte in grigio) in quanto rappresentano un comando non ancora attivo per il progetto in corso. Vediamo ora quali sono i comandi facendo una panoramica dei menu e della barra strumenti dell'Editor di VBA riassumendo i comandi ed elencando lo scopo di ciascuno.

- ***Salva***: Salva il progetto VBA corrente su disco inclusi tutti i moduli e Form
- ***Importa file***: Aggiunge un modulo, Form o classe al progetto esistente. Si possono salvare solo moduli, Form o classi precedentemente salvati da un altro progetto con il comando *Esporta file*
- ***Esporta file***: Salva il modulo, Form o classe corrente in un file di formato testo
- ***Elimina***: Elimina definitivamente dal progetto VBA il modulo o Form selezionato
- ***Stampa***: Stampa un modulo o Form ai fini di documentazione
- ***Chiudi e torna a Excel***: Chiude l'Editor di VB e ritorna ad Excel

Menù modifica

Il menù modifica contiene i comandi per manipolare il codice sorgente di una macro nella finestra del Codice e gli oggetti su un Form

- ***Annulla:*** Annulla il comando più recente
- ***Ripeti***: Ripete il comando annullato per ultimo
- ***Taglia***: Elimina il testo o l'oggetto selezionato nel modulo o nella Form
- ***Copia***: Copia il testo, o l'oggetto, selezionato che viene conservato negli appunti di Windows
- ***Incolla***: Incolla il testo, o l'oggetto, dagli appunti di Windows nel modulo o Form corrente
- ***Cancella***: Elimina il testo, o l'oggetto, selezionato dal modulo o Form
- ***Seleziona tutto***: Seleziona tutto il testo di un modulo, o tutti gli oggetti di una Form
- ***Trova***: Permette di localizzare uno specifico testo in un modulo
- ***Trova successivo***: Ripete l'ultima operazione Trova
- ***Sostituisci***: Permette di localizzare uno specifico testo in un modulo e sostituirlo con un altro
- ***Aumenta rientro***: Aumenta il rientro di una tabulazione
- ***Riduci rientro***: Sposta a sinistra di una tabulazione il testo selezionato
- ***Elenca proprietà/metodi***: Apre un elenco a discesa nella finestra proprietà-metodi del codice che indica tutte le proprietà e metodi dell'oggetto di cui si è appena digitato il nome. Se il cursore si trova

in un punto vuoto della finestra del codice, questo comando apre un elenco di tutte le proprietà e metodi globalmente disponibili

- ***Elenca costanti***: Apre un elenco a discesa nella finestra Codice che mostra le costanti valide della proprietà appena digitata preceduta da un segno di uguale **=**
- ***Informazioni Rapide***: Apre una finestra di aiuto che mostra la sintassi corretta di una procedura, funzione o enunciato appena digitato nella finestra Codice
- ***Informazioni parametri***: Apre una finestra di aiuto che mostra i parametri (argomenti) di una procedura, funzione o enunciato appena digitato nella finestra Codice
- ***Completa parola***: Permette all'editor di VB di completare la parola che si sta digitando
- ***Segnalibri***: Apre un sotto menu di scelte per inserire, eliminare o saltare a un segnalibro inserito nel modulo

Menù Visualizza

Il menu visualizza mette a disposizione i comandi per scegliere quali elementi dell'editor VB si vogliono vedere e come visualizzarli

- ***Codice***: Attiva la finestra del Codice che mostra il sorgente associato al modulo o Form selezionato
- ***Oggetto***: Visualizza l'oggetto attualmente selezionato in Gestione Progetti
- ***Definizione***: Visualizza il codice sorgente della procedura o funzione su cui sta al momento il cursore
- ***Ultima posizione***: Salta all'ultima posizione di un modulo dopo un precedente impiego del comando definizione o dopo una modifica al codice
- ***Visualizzatore oggetti***: Apre il visualizzatore oggetti con cui si può stabilire quali macro sono disponibili al momento
- ***Finestra immediata***: Visualizza la finestra immediata del debugger di VBA
- ***Finestra locali***: Visualizza la finestra Locali del debugger di VBA
- ***Finestra controllo***: Visualizza la finestra Controllo (Espressione di controllo) del debugger di VBA
- ***Stack chiamate***: Visualizza lo stack delle chiamate della procedura o funzione VBA
- ***Gestione progetti***: Visualizza la finestra Gestione Progetti
- ***Finestra proprietà***: Visualizza la finestra delle proprietà
- ***Casella degli strumenti***: Visualizza la casella degli strumenti che può essere utilizzata per aggiungere controlli alle finestre di dialogo
- ***Ordine di tabulazione***: Visualizza la finestra di dialogo Ordina tabulazioni che viene usata per creare finestre personalizzate
- ***Barre degli strumenti***: Mostra un sotto menu per visualizzare o nascondere le diverse barre degli strumenti dell'Editor di VB
- ***Microsoft Excel***: Per tornare ad Excel da cui è stato avviato l'Editor VB

Menù Inserisci

Il menu inserisci permette di aggiungere vari oggetti (moduli, Form) al progetto

- ***Routine***: Inserisce una nuova procedura Sub, Function o Property nel modulo corrente
- ***Userform***: Inserisce una nuova Form al progetto
- ***Modulo***: Aggiunge un nuovo modulo al progetto corrente.
- ***Modulo di classe***: Aggiunge un nuovo modulo di classe al progetto corrente
- ***File***: Permette di inserire in un modulo un file di testo che contiene codice sorgente VBA

Menù Formato

I comandi del menù formato si usano per creare proprie finestre di dialogo personalizzate. Essi permettono di allineare gli oggetti su un Form, di variare la dimensione dei controlli in modo da adattarsi ai contenuti e di fare molte altre cose utili

- ***Allinea*:** Apre un sotto menù di comandi che permettono di allineare gli oggetti selezionati su un Form in diversi modi.
- ***Rendi uguale*:** Apre un sotto menù di comandi che permettono di portare gli oggetti selezionati alla medesima grandezza
- ***Adatta*:** Modifica contemporaneamente altezza e larghezza di un oggetto
- ***Adatta alla griglia*:** Modifica contemporaneamente altezza e larghezza di un oggetto così da adattarlo ai punti più vicini della griglia (quando si progetta una Form l'Editor VB visualizza una griglia o reticolo di punti che aiuta a posizionare gli oggetti sulla Form
- ***Spaziatura orizzontale*:** Apre un sotto menu di comandi che permettono di regolare la spaziatura orizzontale degli oggetti selezionati pareggiandola, riducendola, aumentandola, o eliminando tutti gli spazi orizzontali tra gli oggetti.
- ***Spaziatura verticale*:** Apre un sotto menu di comandi che permettono di regolare la spaziatura verticale degli oggetti selezionati. Si può pareggiare la spaziatura, ridurla, aumentarla o eliminare tutti gli spazi verticali fra gli oggetti
- ***Centra nel form*:** Apre un sotto menu di comandi che permettono di regolare la posizione degli oggetti selezionati centrandoli orizzontalmente o verticalmente nella Form
- ***Disponi pulsanti*:** Apre un sotto menu di comandi che permettono di disporre automaticamente sul form i pulsanti di comando in una riga a spaziatura uniforme lungo il lato basso o destro della Form
- ***Raggruppa*:** Collega assieme in singoli gruppi più oggetti selezionati in modo da poterli spostare, ridimensionare, tagliare o copiare come una singola unità
- ***Annulla raggruppamento*:** Toglie il raggruppamento degli oggetti precedentemente raggruppati col comando raggruppa
- ***Ordina*:** Apre un sotto menu di comandi che permettono di cambiare l'ordine dall'alto-in-basso di oggetti sovrapposti su una Form. Si usa per far si che una casella di testo compaia sempre sopra ad un oggetto grafico su una Form

Menù Debug

I comandi del menu Debug vengono usati quando si fa il collaudo delle macro, ovvero quando si procede al debugging delle stesse, che consiste in un processo atto a individuare gli errori nel programma

- ***Compila*:** Procede alla compilazione del progetto selezionato in gestione progetti
- ***Esegui istruzione*:** Esegue il codice sorgente un enunciato alla volta
- ***Esegui istruzione/routine*:** Simile al precedente, permette di eseguire in una sola volta le istruzioni di una macro senza doverle eseguire passo dopo passo
- ***Esci da istruzione/routine*:** Esegue tutte le istruzioni rimanenti di una macro senza procedere più passo dopo passo
- ***Esegui fino al cursore*:** Esegue tutti gli enunciati del codice sorgente da quello corrente fino alla posizione del cursore
- ***Aggiungi espressione di controllo*:** Permette di specificare variabili o espressioni che contengono valori che si vogliono esaminare mentre il codice sorgente viene eseguito
- ***Modifica espressione di controllo*:** Permette di modificare le specifiche delle variabili o espressioni di controllo create in precedenza con il comando Aggiungi espressione di controllo

- ***Controllo immediato*:** Visualizza il valore corrente di una data espressione di controllo
- *Imposta/rimuovi punto di interruzione*: Evidenzia e contraddistingue un punto del codice sorgente in cui si vuole che l'esecuzione si arresti
- ***Rimuovi punto di interruzione*:** Elimina tutti i punti di interruzione immessi in un modulo
- ***Imposta istruzione successiva*:** Permette di modificare il normale flusso di esecuzione del codice specificando la linea del codice sorgente che verrà eseguita al passo successivo
- ***Mostra istruzione successiva*:** Visualizza la linea del codice sorgente che verrà eseguita successivamente evidenziandola

Menù Esegui

I comandi del menu esegui permettono di avviare l'esecuzione di una macro, interromperla o riprendere l'esecuzione.

- ***Esegui Sub/Userform*:** Con questo comando si esegue la macro in cui si trova il cursore di testo. Se il Form è attivo VBA esegue il Form
- ***Interrompi*:** Interrompe l'esecuzione del codice VBA e apre l'Editor VB in modalità *Interruzione*
- ***Ripristina*:** Annulla i valori di tutte le variabili a livello modulo e lo stack delle chiamate
- ***Esci da modalità progettazione*:** Attiva o disattiva la modalità progettazione di un progetto. Quando è attiva non viene eseguito nessun codice del progetto

Menu Strumenti

I comandi del menu strumenti non solo rendono possibile selezionare la macro da eseguire, ma consentono anche l'accesso a librerie di macro e controlli per i Form addizionali esterni oltre a quelli integrati nel VBA

- ***Riferimenti*:** Visualizza la finestra di dialogo dei Riferimenti con cui si possono stabilire riferimenti a librerie di oggetti, di tipi o a altro progetto VBA
- ***Controlli aggiuntivi*:** Visualizza la finestra di dialogo Controlli aggiuntivi che permette di personalizzare la barra strumenti casella strumenti che serve ad aggiungere controlli ai Form diversi da quelli integrati nel VBA
- ***Macro*:** Visualizza la finestra di dialogo Macro. Con cui si creano, eseguono, modificano o eliminano macro
- ***Opzioni*:** Visualizza la finestra di dialogo Opzioni in cui si possono scegliere varie opzioni per l'Editor VB
- ***Proprietà*:** Visualizza la finestra di dialogo Proprietà progetto in cui si possono impostare varie proprietà del progetto VBA
- ***Firma digitale*:** Visualizza la finestra di dialogo Firma digitale che permette di vedere le informazioni correnti della firma digitale e di "firmare" il proprio progetto VBA con un certificato di esistenza di firma

L'Editor VB dispone di altri tre menu: **Aggiunte, Finestra e Guida.**

Il menu **Aggiunte** contiene una sola voce "*Gestione delle aggiunte*". Questo comando apre la corrispondente finestra di dialogo Gestione delle aggiunte che può essere impiegata per aggiungere o eliminare programmi aggiuntivi (Add-In) di Visual Basic.

Il menu **Finestra** consente di passare da un Form all'altro, portando in primo piano sul desktop quello su cui vogliamo lavorare
Il menu **Guida** dà accesso alla guida on-line di Visual Basic For Applications che contiene istruzioni, suggerimenti, spiegazioni tecniche consultabili durante il lavoro di programmazione. Per una consultazione rapida, basta posizionare il mouse sull'oggetto che interessa e poi premere il tasto F1.

La Barra strumenti dell'Editor di VBA

La selezione di un pulsante di comando col mouse risulta in genere più comoda per l'utente rispetto alla scelta di un comando da menu. L'Editor VB presenta i comandi più importanti e più usati sotto forma di pulsanti (icone) in una barra strumenti. Per default l'Editor VB visualizza solo la barra strumenti Standard come si vede in **figura 2**

Fig. 2

Oltre a questa l'Editor di VB dispone di altre tre barre strumenti: **Modifica, Debug e Userform.** La barra strumenti *Modifica* contiene vari pulsanti di comando utili per editare il testo nella finestra del codice. Si può controllare quali barre strumenti vengono visualizzate tramite il comando **Visualizza - Barre strumenti** e dato che per default l'Editor di VB non visualizza la barra strumenti Modifica, sarà necessario renderla visibile manualmente. Per visualizzare o nascondere una delle barre strumenti dell'Editor si procede così: Selezionare il comando **Visualizza - Barre strumenti** per aprire un sotto menu che elenca le varie barre strumenti dell'Editor. Fare clic sul nome della barra strumenti che si vuole visualizzare, per esempio se si vuole rendere visibile la barra strumenti Modifica, fare clic nel sotto menu su Modifica e l'Editor di VB renderà visibile la barra strumenti selezionata

Per default l'Editor di VB mostra la barra *Strumenti Standard* ancorata in alto nella finestra dell'Editor. Una barra strumenti può anche essere resa *flottante*, ossia non ancorata ad un lato della finestra. In tal caso essa compare in una finestra dotata di bordi e titolo.

Fig. 3

Per ancorare o rendere flottante una barra strumenti nell'Editor di VB si usa la medesima tecnica usata per le barre strumenti di Excel, basta trascinare la barra strumenti nella posizione desiderata e poi rilasciare il pulsante del mouse. La barra strumenti Standard dell'Editor di VB contiene 18 pulsanti (icone) ognuno dei quali offre l'accesso rapido a un comando di menu con un semplice clic. I comandi della barra Strumenti Standard *partendo dal lato sinistro di* ***figura 2*** sono:

- ***Visualizza***: Passa all'applicazione dalla quale è stato avviato l'Editor, nel nostro caso Excel
- ***Inserisci oggetto***: Facendo clic sulla freccina in giù sulla destra viene visualizzato un elenco degli oggetti che si possono inserire nel progetto: Userform, Modulo, Modulo di classe o Procedura
- ***Salva***: Salva il progetto corrente
- ***Taglia***: Taglia il testo o l'oggetto selezionato e lo trasferisce negli appunti di Windows
- ***Copia***: Copia il testo o l'oggetto selezionato e lo trasferisce negli appunti di Windows
- ***Incolla***: Incolla il testo o l'oggetto contenuto negli appunti nella finestra del Codice o nella Userform nella posizione attuale del cursore
- ***Trova***: Apre la finestra di dialogo Trova per individuare la posizione di una specifica parola o frase in un modulo
- ***Annulla***: Annulla l'ultimo comando immesso
- ***Ripeti***: Ripete l'ultimo comando immesso
- ***Esegui***: Esegue la procedura corrente o form
- ***Interrompi***: Interrompe l'esecuzione del codice VBA
- ***Ripristina***: Ripristina il codice VBA nello stato iniziale
- ***Modalità Progetto***: Fa passare nella modalità Progetto del VBA
- ***Gestione Progetto***: Apre la finestra Gestione Progetti
- ***Finestra Proprietà***: Apre la finestra delle Proprietà
- ***Visualizzatore Oggetti***: Apre la finestra di dialogo del Visualizzatore degli Oggetti
- ***Casella strumenti***: Visualizza la barra strumenti Casella Strumenti
- ***Assistente di office***: Visualizza l'assistente di Office e offre una guida

Impostazioni delle proprietà dei controlli

Mentre viene sviluppata un'applicazione in VBE (ambiente Visual Basic) viene definita come **fase di progettazione** (*Design Time*) il momento in cui si creano Form, si aggiungono controlli e si impostano le proprietà dei vari oggetti, mentre invece si definisce fase di **esecuzione** (*Run-Time*) il momento in cui il codice viene eseguito e l'applicazione è in esecuzione.

Durante il periodo di Run-Time lo sviluppatore interagisce con l'applicazione, proprio come un utente e il codice non può essere manipolato, mentre eventuali manipolazioni nella fase di progettazione non sono permanenti. Per esempio se si aggiunge un controllo *CheckBox* nel codice utilizzando il metodo*Add Method [Set ctrl = Controls.Add("Forms.CheckBox.1")]* il controllo apparirà una volta che si manda in esecuzione la UserForm, ma quando si termina l'esecuzione e si torna al VBE, l'oggetto CheckBox non è presente, allo stesso modo, se si imposta la *Caption* di un controllo OptionButton nel codice, viene visualizzato quando viene eseguita la UserForm, ma tornerà al suo aspetto originale quando si termina l'esecuzione. In sostanza la struttura *ControlTipText*si trova in fase di progettazione, ma è visibile sul controllo unicamente durante la fase di Run-Time.

Impostazione delle proprietà dei controlli

Se il codice è in una procedura nel modulo di codice della Form, si può utilizzare la sintassi: *Controlname.Property = Setting/Value*, ma se il codice è in un modulo standard o nel modulo di codice di un Form diverso, la sintassi diventa: *UserFormName.Controlname.Property = Setting/Value*.
Alcuni *Esempi di sintassi*

```
Label1.Font.Name = "Arial"
Label1.ForeColor = RGB (255, 255, 0)
OptionButton1.BackColor = 255
CheckBox1.Value = false
CheckBox1.Alignment = fmAlignmentLeft
TextBox1.MultiLine = True
TextBox1.WordWrap = True
TextBox1.ScrollBars = 2
OptionButton1.AutoSize = True
Me.TextBox1.Enabled = False
TextBox1.TextAlign = fmTextAlignLeft
TextBox1.Text = "Ciao"
CommandButton1.Left = 50
TextBox1.MaxLength = 5
```

Proprietà Name

Si può utilizzare la proprietà *Name* per specificare un nome per un controllo o per specificare il nome del font del carattere utilizzato nella parte di testo di un controllo. La proprietà Name in una Form può essere impostata solo in fase di progettazione e non può essere impostata in fase di esecuzione, inoltre per i controlli può essere impostata sia in fase di progettazione o in fase di esecuzione, ma se si aggiunge un controllo in fase di progettazione, il suo nome non può essere modificato in fase di esecuzione. In genere, il nome predefinito del primo CheckBox creato è CheckBox1, il nome predefinito del secondo CheckBox sarà CheckBox2, e così via, anche per gli altri controlli.

E' possibile modificare il nome di un controllo cliccando su "*Name*" nella finestra delle Proprietà, ricordando che il nome deve iniziare con una lettera, può avere qualsiasi combinazione di lettere, numeri o underscore, non può avere spazi o simboli e può avere una lunghezza massima di 40 caratteri.

Potrebbe essere una buona idea utilizzare un prefisso di 3 lettere in minuscolo, per individuare il rispettivo controllo, e i caratteri che seguono il prefisso possono essere caratteristici per una più facile leggibilità. I prefissi comunemente utilizzati per diversi controlli sono: frm per UserForm; LBL per etichette; txt per TextBox; CMB per ComboBox; lst per ListBox; chk per CheckBox; opt per OptionButton; Fra per Frame; cmd per CommandButton; TBS per TabStrip; RFE per RefEdit; e così via.

La Proprietà Caption

Caption è il testo che descrive e identifica una Form o un controllo e verrà visualizzato nell'intestazione della Form, o di qualsiasi altro controllo, e può essere impostata nella finestra Proprietà o con il codice usando questa Sintassi: *object.Caption = String*

Le Proprietà Height e Width

L'altezza e la larghezza viene misurata in punti e queste proprietà sono applicabili sia a un oggetto Form che a tutti i controlli disponibili. È possibile inserire manualmente l'altezza e la larghezza nella finestra Proprietà, e per queste proprietà, VBA accetta solo valori che sono maggiori o uguali a zero, e possono essere impostate nella finestra Proprietà o con il codice VBA con la sintassi: *object.Height = Number Object.Width = Number*. È inoltre possibile ridimensionare un controllo manualmente con il mouse, quando viene selezionato il puntatore del mouse cambierà aspetto presentandosi come una freccia a due punte e il controllo mostrerà le maniglie di regolazione che sono situate negli angoli e a metà del controllo stesso e posizionando il cursore su una qualsiasi di queste maniglie e cliccando su di essa si possono modificare le dimensioni del controllo trascinando il cursore per portarlo alla dimensione desiderata, che una volta raggiunta si rilascia il pulsante del mouse

Le Proprietà Left e Top

La proprietà *Left* imposta la distanza tra il bordo sinistro del controllo e il bordo sinistro della Form che lo contiene, mentre invece la proprietà *Top* imposta la distanza tra il bordo superiore del controllo e il bordo superiore della Form e per entrambi i controlli la distanza è impostata in pixel, inoltre è possibile inserire manualmente le proprietà Left e Top nella finestra Proprietà tenendo presente che queste proprietà sono applicabili a tutti i controlli. Se il valore di Left o Top è impostato a zero, il controllo apparirà sul bordo sinistro o il bordo superiore della Form che lo contiene, e specificando un valore minore di zero in una di queste proprietà verrà tagliata una porzione del controllo riducendone la visibilità nel modulo. Queste proprietà possono essere impostate con il codice VBA con questa Sintassi: *object.Left = Number object.Top = Number* .

La Proprietà Value

Questa proprietà determina lo stato di selezione di un controllo o specifica il contenuto dello stesso ed è applicabile a tutti i controlli tranne Label, Frame e Image. Per quanto riguarda i controlli CheckBox, OptionButton e ToggleButton, impostando un valore di -1 (equivale a True), indica che il controllo è selezionato, un valore 0 (equivale a False), indica che il controllo è deselezionato e un valore Null indica che il controllo non è né selezionato né cancellato, e in questo caso apparirà ombreggiato. Per i controlli ScrollBar e SpinButton, la proprietà *Value* indica il loro valore attuale, che è un numero intero compreso tra il valore minimo e massimo specificati nelle proprietà *Max* e *Min*. Per i controlli ComboBox e ListBox (Value non può essere utilizzato con un ListBox a selezione multipla), rappresenta il valore della cella attualmente selezionata, mentre per un controllo CommandButton, equivale a un valore booleano (True o False) che indica se è stato scelto il comando e l'impostazione predefinita è False e se è impostato a True (può essere fatto solo con il codice VBA) richiamerà l'evento *Click* del pulsante. Per un controllo *Multipage*, la proprietà Value è impostata solo con il codice VBA ed è rappresentata da un numero Intero che indica se la pagina corrente è attiva, ricordando che le pagine sono numerate a partire da zero (0).

Per un controllo TextBox, si riferisce al testo nella casella di testo e la proprietà Value può essere impostata nella finestra Proprietà (fatta eccezione per controlli CommandButton e Multipage) o con il codice VBA. Sintassi: *Object.Value = Variant*

La Proprietà Accelerator

Questa proprietà è applicabile ai controlli Label, CheckBox, OptionButton, ToggleButton, CommandButton e Multipage, imposta la chiave per accedere a un controllo, ed è indicato come la chiave di accesso (o tasto di scelta rapido) che è costituita da un singolo carattere, che premuto in combinazione con e dopo il tasto *Alt* è usato come scorciatoia e se usato per un controllo avvia l'evento *Click*. Per fare clic su un pulsante di comando in una Form, il tasto di scelta rapida può essere impostato come lettera "E" e premendo *Alt + E* si avvia l'evento click. Nel caso in cui *Accelerator* è impostato per una Label, il controllo successivo che segue la Label nell'ordine di tabulazione riceve il *Focus* (ma non l'esecuzione dell'evento Click). Inoltre si tenga presente che Il carattere utilizzato come valore Accelerator è *key-sensitive*, il che significa che l'impostazione della chiave avviene come lettera **P** equivale anche alla lettera **p**perché vengono inserite premendo lo stesso tasto.

La Proprietà Alignment

Questa proprietà è applicabile ai controlli CheckBox, OptionButton e ToggleButton e specifica come una Caption apparirà rispetto al controllo. Ci sono due impostazioni:

- fmAlignmentLeft (valore 0) – La Caption appare a sinistra del controllo
- fmAlignmentRight (valore 1) - Questa è l'impostazione di default in cui la Caption viene visualizzata a destra del controllo.

Si tenga presente che il controllo ToggleButton ha *Alignment* come una delle sue proprietà, ma è disabilitato e non può essere specificato per questo controllo e Il testo della Caption è sempre allineato a sinistra.

La Proprietà AutoSize

Questa proprietà è applicabile ai controlli Label, TextBox, ComboBox, CheckBox, OptionButton, ToggleButton, CommandButton, Image e RafEdit ed è rappresentata da un valore booleano (True o False) che specifica se il contenuto da visualizzare del controllo viene ridimensionato automaticamente oppure no. Se *AutoSize* viene posta a TRUE, si ridimensiona automaticamente il controllo, mentre impostandola a FALSE (opzione predefinita) mantiene la dimensione del controllo e se il contenuto supera l'area del controllo viene tagliato. Per i controlli TextBox e ComboBox, *AutoSize* si applica al testo visualizzato, mentre per il controllo Image, AutoSize vale per l'immagine visualizzata, mentre per altri controlli si applica alla Caption, mentre le impostazioni per il controllo TextBox sono:

- Se il TextBox è a linea singola, AutoSize ridimensiona la larghezza del TextBox alla lunghezza del testo
- Se il TextBox è MultiLinea, senza testo, AutoSize ridimensiona la larghezza per visualizzare una singola lettera e ridimensiona l'altezza per visualizzare l'intero testo
- Se la TextBox è MultiLine con il testo, AutoSize non cambia la larghezza del TextBox e ridimensiona l'altezza per visualizzare l'intero testo.

La Proprietà BackColor

Questa Proprietà si applica a tutti i controlli e Form ed imposta il colore di sfondo. Per il controllo Multipage la proprietà può essere impostata solo con il codice VBA

La Proprietà BackStyle

BackStyle è applicabile ai controlli Label, TextBox, ComboBox, CheckBox, OptionButton, ToggleButton, CommandButton, Image e RafEdit e determina se lo sfondo dei controlli sarà opaco o trasparente. Ha due impostazioni:

- fmBackStyleTransparent (valore 0) per sfondo trasparente, in cui tutto lo sfondo del controllo è visibile
- fmBackStyleOpaque (valore 1) per lo fondo opaco, in cui nulla è visibile sullo sfondo del controllo e questo è anche il default.

Se la proprietà BackStyle (per i controlli) è impostato su *fmBackStyleOpaque* la proprietà *BackColor* non avrà nessun effetto.

La Proprietà BorderColor

Questa proprietà è applicabile alle Form e ai controlli Label, TextBox, ComboBox, ListBox, Frame, Image e RafEdit e imposta il colore del bordo. Se nella proprietà *BorderStyle* si imposta il valore *fmBorderStyleNone*, questa proprietà non avrà alcun effetto, in quanto la proprietà *BorderStyle* definisce i colori del bordo utilizzando la proprietà BorderColor , mentre la struttura *SpecialEffect* utilizza esclusivamente colori di sistema (che fanno parte del Pannello di controllo di Windows) per definire i colori del bordo.

La Proprietà BorderStyle

Questa proprietà è applicabile alle Form e controlli Label, TextBox, ComboBox, ListBox, Frame, Image e RafEdit e specifica il tipo di bordo per un oggetto (controllo o Form). Ha due impostazioni:

- fmBorderStyleNone (valore 0) per nessun bordo
- fmBorderStyleSingle (valore 1) per un bordo a linea singola.

Da ricordare che Form, Label, TextBox, ComboBox, ListBox e Frame hanno il valore di default pari a 0, mentre il valore predefinito per un'immagine è 1.*BorderStyle* definisce i colori dei bordi con la proprietà BorderColor e non è possibile utilizzare contemporaneamente BorderStyle e SpecialEffect per specificare il bordo di un controllo, inoltre se la proprietà *SpecialEffect* per un Frame è impostata a zero, la proprietà *BorderStyle* viene ignorata.

La Proprietà ControlSource

Questa proprietà è applicabile ai controlli TextBox, ComboBox, ListBox, CheckBox, OptionButton, ToggleButton, ScrollBar e SpinButton e corrisponde a una cella o un campo (intervallo di celle) che viene utilizzato per impostare o conservare la proprietà *Value* di un controllo. Cambiando il valore del controllo si aggiorna automaticamente la cella collegata e un cambiamento nella cella collegata aggiornerà il valore del controllo. Se nella cella A1 viene inserita la proprietà *ControlSource* di un CheckBox, e se la cella A1 in ActiveSheet contiene TRUE, il CheckBox apparirà selezionato all'attivazione del modulo e se si deseleziona l'opzione, la cella A1 cambierà il suo contenuto in FALSE. In una ListBox in cui il ControlSource menziona *Foglio1! D2*, il valore nella BoundColumn della riga selezionata vengono memorizzati nella cella D2 del Foglio1, mentre invece in una TextBox in cui il ControlSource menziona *Foglio3! F2*, il testo o il valore nella TextBox vengono memorizzati nella cella F2 del Foglio3 e se la cella F2 non contiene nessun testo, questo apparirà nella TextBox all'attivazione della Form. Il valore predefinito è una stringa vuota che indica che la proprietà *ControlSource* non è stata impostata.

La Proprietà ControlTipText

E' applicabile a tutti i controlli e specifica il testo visualizzato quando l'utente posiziona il mouse su un controllo. E utile nel dare consigli o chiarimenti per l'utente sull'utilizzo del controllo. Il valore predefinito è una stringa vuota che indica che non verrà visualizzato alcun testo.

La Proprietà Enabled

Questa Proprietà si applica a tutti i controlli e Form e rappresenta un valore booleano (True o False) che specifica se il controllo è attivo e può rispondere a eventi generati dall'utente, (cioè l'utente può interagire con il controllo tramite mouse, i tasti o tasti di scelta rapida). Il valore predefinito è True, che indica che il controllo è attivo, mentre un valore False indica che l'utente non può interagire con il controllo. Il controllo è generalmente accessibile tramite un codice anche nel caso il valore sia impostato su False. Se *Enabled* è impostato su False, il controllo è inattivo (tranne che per le immagini), mentre se Enabled è impostata su false per un Form o un Frame, tutti i controlli che contengono sono disabilitato. La proprietà *Enabled* di una TextBox è particolarmente utile quando non si desidera consentire all'utente di digitare direttamente nella casella di testo, ma deve essere riempito solo tramite la selezione effettuata dall'utente in altro controllo, ad esempio da un ListBox.

La Proprietà Locked

Questa proprietà è applicabile ai controlli TextBox, ComboBox, ListBox, CheckBox, OptionButton, ToggleButton, CommandButton e RafEdit e rappresenta un valore booleano (True o False) che specificare se il controllo è modificabile o meno. Il valore TRUE indica che non è modificabile, mentre il valore predefinito è False in cui il controllo può essere modificato.

Alcuni esempi di utilizzo delle proprietà Enabled e Locked in combinazione:

- Se *Enabled* è True e *Locked* è False: il controllo risponde agli eventi generati dall'utente e appare normalmente, i dati possono essere copiati e modificati nel controllo.
- Se *Enabled* è True e *Locked* è True, il controllo risponde agli eventi generati dall'utente e appare normalmente, i dati possono essere copiati, ma non modificati nel controllo.
- Se *Enabled* è False (indipendentemente dal valore di *Locked*), il controllo non può rispondere a eventi generati dall'utente e viene visualizzato in grigio, i dati non possono essere né copiati né modificati nel controllo.

Oggetto Font

Questa proprietà si applica a Form e a tutti i controlli tranne ScrollBar, SpinButton e Image e determina il tipo di carattere utilizzato in un controllo o Form. È possibile specificare il nome del font, impostare lo stile del carattere (normale, corsivo, grassetto, etc.), o sottolineare il testo barrato, e regolare la dimensione del carattere. Per i controlli TextBox, ComboBox e ListBox, il tipo di carattere del testo visualizzato è impostato, mentre per altri controlli è impostato il carattere della Caption. L'impostazione *Font* di una Form imposta automaticamente il carattere di tutti i controlli, se posta all'inizializzazione della Form, ma non modifica il tipo di carattere di questi controlli se fossero già presenti per i quali sarà necessario reimpostare il carattere di ogni singolo controllo separatamente. La proprietà Font per un controllo Multipage può essere utilizzata solo con il codice VBA.

La Proprietà ForeColor

Questa proprietà si applica a Form e a tutti i controlli tranne Image e specifica il colore di primo piano, cioè il colore del testo visualizzato. Per quanto riguarda i controlli Font, *ForeColor* determina il colore del testo, mentre in un Frame, ForeColor determina il colore della Caption e in una ScrollBar o SpinButton, ForeColor determina il colore delle frecce. Da notare che l'impostazione ForeColor di una Form imposta automaticamente la proprietà ForeColor dei controlli Label, CheckBox, OptionButton, Frame, Multipage e TabStrip se inseriti nell'inizializzazione della Form stessa, ma non cambierà la proprietà ForeColor di questi controlli se fossero già presenti, per i quali sarà necessario reimpostarla per ogni singolo controllo separatamente. La proprietà ForeColor per un controllo Multipage può essere utilizzata solo con il codice VBA.

La Proprietà MouseIcon

Questa proprietà è applicabile a Form e a tutti i controlli, ad eccezione di Multipage e assegna un'immagine a un controllo che viene visualizzato quando l'utente sposta il mouse su tale controllo. Image viene assegnato specificando il percorso e il nome del file dove è collocata l'immagine e per utilizzare la struttura *MouseIcon* è necessario che la proprietà *MousePointer* sia impostata su *fmMousePointerCustom* (valore 99).

La Proprietà MousePointer

Questa proprietà è applicabile a Form e a tutti i controlli, ad eccezione di Multipage e specifica che tipo di puntatore del mouse sarà visibile quando l'utente sposta il mouse sopra un controllo. Ci sono 15 impostazioni:

- fmMousePointerDefault (valore 0) - puntatore standard, è il valore di default
- fmMousePointerArrow (valore 1) - freccia
- fmMousePointerCross (valore 2) - puntatore a croce
- fmMousePointerIBeam (valore 3) - I-Beam
- fmMousePointerSizeNESW (valore 6) - freccia a due punte punta nord-est e sud-ovest
- fmMousePointerSizeNS (valore 7) - freccia a due punte punta nord e sud
- fmMousePointerSizeNWSE (valore 8) - freccia a due punte punta nord-ovest e sud-est
- fmMousePointerSizeWE (valore 9) - doppia freccia che punta a ovest e ad est
- fmMousePointerUpArrow (valore 10) - freccia
- fmMousePointerHourglass (valore 11) - clessidra
- fmMousePointerNoDrop (valore 12) - cerchio con una linea diagonale, che appare come un simbolo "Not", che indica un controllo non valido
- fmMousePointerAppStarting (valore 13) - freccia e clessidra
- fmMousePointerHelp (valore 14) - freccia e il punto interrogativo
- fmMousePointerSizeAll (valore 15) - freccia a quattro punte, rivolto verso nord, sud, est e ovest
- fmMousePointerCustom (valore 99) - l'immagine specificata dalla struttura MouseIcon

La Proprietà Picture

Questa proprietà è applicabile a Form e controlli Label, CheckBox, OptionButton, ToggleButton, Frame, CommandButton, Multipage e Image e specifica l'immagine da visualizzare in un controllo, specificando il percorso e il nome del file, per rimuovere l'immagine si deve premere CANC sul valore della proprietà. Per i controlli con Caption, è possibile specificare la posizione dell'immagine utilizzando la proprietà *PicturePosition* mentre per altri controlli e Form, si deve utilizzare la proprietà *PictureAlignment* per specificare la posizione dell'immagine e utilizzare la proprietà *PictureSizeMode* per specificare la modalità (dimensioni, scala, etc.) per visualizzare l'immagine.

La Proprietà PicturePosition

Questa proprietà è applicabile ai controlli Label, CheckBox, OptionButton, ToggleButton e CommandButton e specifica dove deve comparire l'immagine nel controllo. Ci sono 3 impostazioni in un formato in cui la stringa *fmPicturePosition* è seguita dalla posizione dell'immagine rispetto alla sua voce e al successivo allineamento della Caption relativa all'immagine, cioè.

- fmPicturePositionLeftTop - l'immagine viene visualizzata a sinistra del titolo e la Caption è allineata con la parte superiore del quadro
- fmPicturePositionCenter - Sia l'immagine che la Caption sono concentrati nel controllo e la Caption è sulla parte superiore del quadro.
- fmPicturePositionAboveCenter (valore 7) – E' il valore predefinito e l'immagine appare sopra la Caption e la stessa è centrata sotto l'immagine.

La Proprietà SpecialEffect

Questa proprietà è applicabile a Form e controlli Label, TextBox, ComboBox, ListBox, CheckBox, OptionButton, ToggleButton, Frame, Image e RafEdit e determina come appare visivamente il controllo. Per un CheckBox, OptionButton, o ToggleButton, le due impostazioni sono:

- fmButtonEffectFlat (valore 0)
- fmButtonEffectSunken (valore 2) - di default per CheckBox e OptionButton.

Per altri controlli sono applicabili cinque impostazioni che sono:

- fmSpecialEffectFlat (valore 0) - default per Form e i controlli Image e Label
- fmSpecialEffectRaised (valore 1)
- fmSpecialEffectSunken (valore 2) - di default per i controlli TextBox, ComboBox e ListBox
- fmSpecialEffectEtched (valore 3) - di default per Frame
- fmSpecialEffectBump (valore 6).

L'aspetto visivo di ogni impostazione è auto-esplicativo e può essere piatto, in rilievo, incassato, inciso e Bump.
Anche se ToggleButton ha *SpecialEffect* come una delle sue proprietà, è disabilitato e non può essere specificato, inoltre non è possibile utilizzare contemporaneamente sia la proprietà BorderStyle e SpecialEffect per specificare il bordo per un controllo in quanto per uno dei due con valore diverso da zero imposterà automaticamente l'altra proprietà a zero. Se la proprietà SpecialEffect per un Frame è impostato a zero, la proprietà BorderStyle viene ignorata.

La Proprietà TabIndex

Questa proprietà è applicabile a tutti i controlli tranne Image e *TabIndex* rappresenta la posizione del controllo nell'ordine di tabulazione di un Form quando l'utente preme il tasto Tab. Il valore di indice è espresso come un valore Integer, indicando con 0 la prima posizione nell'ordine di tabulazione e il valore più alto dell'Indice sarà uno in meno del numero di controlli nella Form, a cui la proprietà TabIndex è applicabile. L'immissione di un valore di indice inferiore a zero darà un errore e un valore superiore al più alto possibile resetterà al valore più alto, e ogni controllo avrà un valore di indice univoco.

La Proprietà TabStop

Questa proprietà è applicabile a tutti i controlli tranne Label e Image ed è rappresentata da un valore booleano (True o False) che specifica se il controllo può essere selezionato con il tasto Tab. Il valore True è di default, e imposta il controllo come una tabulazione, mentre invece il valore False ignora il controllo, ma la sua posizione nell'ordine di tabulazione (come specificato nella proprietà TabIndex) rimane intatto.

La Proprietà Visible

Questa Proprietà si applica a tutti i controlli e rappresenta un valore booleano (True o False) che è impostato per visualizzare o nascondere un controllo. Il valore di default è True, in cui il controllo è visibile. Questa proprietà è particolarmente utile in cui in seguito a una condizione è possibile attivare un controllo nascosto che altrimenti non si potrebbe vedere nella Form.

La Proprietà WordWrap

Questa proprietà è applicabile ai controlli Label, TextBox, CheckBox, OptionButton, ToggleButton, CommandButton e RafEdit ed è rappresentata da un valore booleano (True o False) che specifica se il testo di un controllo andrà a capo alla riga successiva. Il valore di default è True e se la proprietà *MultiLine* di un controllo è impostata su False, WordWrap viene ignorato nei controlli che supportano entrambe queste proprietà vale a dire TextBox.

Le Funzioni Empty – ZLS – Null – Nothing e Missing

In Excel VBA spesso ci riferiamo a una variabile *Empty* (vuota), a una *ZLS* (stringa di lunghezza zero) o a una stringa nulla (*vbNullString*), a un valore *Null* o a un argomento mancante (*Missing*) o utilizzando la parola chiave *Nothing* (Niente) con una variabile oggetto. E 'importante differenziare e comprendere questi termini ed espressioni mentre vengono utilizzati nel codice VBA. Vediamo ora di comprendere come utilizzare la funzione *VarType* per determinare il sottotipo di una variabile, utilizzando le funzioni *IsEmpty* e *IsNull* per verificare la presenza di valori vuoti, e la funzione *IsMissing* per verificare se gli argomenti opzionali sono stati elencati nella procedura.

La Funzione Empty

Quando si dichiara una variabile utilizzando un'istruzione Dim, si sta riservando la parte sufficiente di memoria per alloccare la variabile nel sistema, (cioè 2 byte per una variabile booleana o Integer, 4 byte per una variabile Long, e così via), e inoltre ci si deve accertare che le informazioni che saranno memorizzate nella variabile abbiano un intervallo consentito (True o False per una variabile booleana, un numero intero compreso tra -32.768 e 32.767 per una variabile Integer, un numero intero compreso tra -2.147.483.648 a 2.147.483.647 per una variabile Long, e così via).

Quando in una dichiarazione di una variabile non si specifica il tipo di dati, oppure se non viene dichiarata, assumerà per impostazione predefinita la forma di tipo **Variant** e può contenere qualsiasi tipo di dati (stringa, data, ora, booleano o valori numerici) e sarà in grado di convertire automaticamente i valori che contiene. Tuttavia, lo svantaggio di questa assegnazione è che deve essere riservata più memoria di quanto richiesto (almeno 16 byte), oltre al fatto che in caso di un errore di digitazione di un nome di variabile non saremmo in grado di riconoscerlo, vale a dire che è possibile digitare rowNumbre invece di rowNumber.

Quando si esegue una macro, tutte le variabili vengono inizializzate ad un valore predefinito e Il valore di default iniziale per una variabile numerica è zero, per una stringa di lunghezza variabile è una lunghezza zero o stringa vuota (""), una stringa di lunghezza fissa viene inizializzata con il codice ASCII 0, o Chr (0), una variabile oggetto su Nothing e una variabile Variant viene inizializzata a vuoto. Nel contesto numerico, una variabile vuota indica uno zero, mentre in un contesto di una variabile stringa vuota è una stringa di lunghezza zero ("") che è indicata anche come una stringa nulla. Tuttavia, si consiglia di specificare esplicitamente un valore iniziale per una variabile, invece di basarsi sul suo valore iniziale di default.

La Funzione IsEmpty

Si può utilizzare la funzione **IsEmpty** per controllare se una variabile è stata inizializzata, in questo caso la funzione restituisce un valore booleano, restituisce True per una variabile non inizializzata o se una variabile è impostata in modo esplicito a Empty, altrimenti la funzione restituisce False. La sintassi è la seguente: *IsEmpty (espressione)*, dove *espressione* è una variabile di tipo Variant che si desidera controllare.

La Funzione ZLS (stringa di lunghezza zero) o vbNullString

ZLS significa stringa di lunghezza zero (""), ed è indicata anche come una stringa nulla, e ha una lunghezza pari a zero (0). Per tutti gli scopi pratici si può utilizzare la costante **vbNullString** che è equivalente a una stringa di lunghezza zero (""), perché VBA interpreta in un modo simile, anche se entrambi non sono in realtà la stessa cosa, in quanto una stringa di lunghezza zero significa in realtà la creazione di una stringa senza caratteri, mentre vbNullString è una costante utilizzata per un puntatore nullo il che significa che nessuna stringa viene creata ed è anche più efficiente o più veloce da eseguire rispetto a ZLS. È possibile usare "" o vbNullString in alternativa nel codice ed entrambi si comportano allo stesso modo, si noti che non vi è alcuna parola chiave Empty in VBA, ma possiamo fare riferimento a "celle vuote "o "celle vuote nel foglio di calcolo Excel".

La Funzione VarType

Si utilizza la funzione **VarType** per determinare il tipo di variabile con la seguente sintassi: *VarType (nome_variabile)* e restituisce un intero che indica il sottotipo della variabile. L'espressione nome_variabile può essere qualsiasi variabile, tranne un tipo di dati definito dall'utente utilizzando l'istruzione Type. Esempi di valori di ritorno sono:

Il valore 0 (costante VarType - vbEmpty, non inizializzato di default), *il valore 1* (costante VarType - vbNull, non contiene dati validi), *il valore 2* (costante VarType - vbInteger, Integer), *il valore 3* (costante VarType - vbLong, Intero long), e così via. Le costanti VarType possono essere utilizzate ovunque nel codice al posto dei valori effettivi.

Esempio: Rappresentare una variabile vuota:

```
Sub Prova1 ()
'la variabile Var1 non è stata dichiarata, quindi è di tipo Variant
'e restituisce 0, che indica come sottotipo una variabile vuota
MsgBox VarType (var1)
'restituisce True, che indica una variabile vuota
MsgBox IsEmpty (var1)
'restituisce False, è una variabile vuota, non una variabile Null
MsgBox IsNull (var1)
'una variabile vuota o uguale a zero in VBA viene rappresentata sia come uno zero
`che come una stringa di lunghezza zero, e restituisce entrambi i messaggi
If var1 = 0 Then
MsgBox "Variabile vuota rappresentata come Zero"
End If
If var1 = "" Then
MsgBox "Variabile vuota rappresentata come Zero-Length (Null) String"
End If
End Sub
```

Esempio: Test per variabili vuote

```
Sub Prova2 ()
Dim var1 As Variant
'variabile non inizializzata, restituisce 0, che indica una variabile vuota
MsgBox VarType (var1)
'restituisce True, indicando una variabile vuota
MsgBox IsEmpty (var1)
'Inizializzare la variabile come stringa
var1 = "Ciao"
`restituisce 8, che indica una variabile Stringa
MsgBox VarType (var1)
'restituisce False, che indica che la variabile non è vuota
MsgBox IsEmpty (var1)
'si imposta la variabile vuota
var1 = Empty
'restituisce 0, che indica variabile vuota
MsgBox VarType (var1)
'restituisce True, che indica variabile vuota
MsgBox IsEmpty (var1)
'Restituisce True per una cella del foglio di lavoro vuota, altrimenti False
MsgBox IsEmpty (ActiveCell)
End Sub
```

Esempio: Inizializzare una variabile Variant

```
Sub Prova3 ()
Dim var1 As Variant
'variabile inizializzata con una stringa di lunghezza zero ("")
var1 = ""
'restituisce False, che indica che la variabile non è vuota
MsgBox IsEmpty (var1)
'restituisce 8, che indica una variabile Stringa
MsgBox VarType (var1)
If var1 = "" Then
MsgBox "Il valore della variabile è una stringa di lunghezza zero"
Else
MsgBox "Il valore della variabile NON è una stringa di lunghezza zero"
End If
If var1 = 0 Then
MsgBox "Il valore della variabile è zero"
Else
MsgBox "Il valore della variabile non è zero"
End If
End Sub
```

Esempio: Controllare una stringa di lunghezza zero:

```
Sub Prova4 ()
Dim var1 As Variant
'variabile non inizializzata, restituisce 0, che indica una variabile vuota
'è rappresentata sia come zero (0) che con una lunghezza zero (Null)
MsgBox VarType (var1)
If var1 = "" Then
MsgBox "True"
End If
If var1 = vbNullString Then
MsgBox "True"
End If
If Len (var1) = 0 Then
MsgBox "True"
End If
End Sub
```

La funzione Null

In VBA, la parola chiave **Null** viene utilizzata per indicare che una variabile non contiene dati validi e il valore che indica una variabile che non contiene dati validi il risultato è Null se:

- Si assegna esplicitamente Null a una variabile
- Se si eseguono operazioni tra espressioni che contengono la parola chiave Null

La parola chiave Null viene utilizzata per variabili di tipo Variant, e solo una variabile Variant può essere Null, mentre variabili di qualsiasi altro tipo rimanderanno un errore, inoltre una variabile Null non è da intendere come una stringa di lunghezza zero (""), e non è vuota, ma indica una variabile non ancora inizializzata. Se si tenta di ottenere il valore di una variabile Null o un'espressione che è Null, si otterrà un *errore 94 di Utilizzo non valido di Null*.

La Funzione IsNull

La funzione **IsNull** restituisce un valore booleano, dove True rappresenta un'espressione che Null (non contiene dati validi), mentre False indica un'espressione che contiene dati validi. La sintassi è la seguente: *IsNull (espressione)* e l'argomento *espressione* è una variante che contiene un valore numerico o stringa.

Esempio: Variabile Integer

```
Sub Prova5 ()
Dim intVar As Integer 'nessun valore iniziale è assegnato alla variabile Integer
MsgBox IsNull (intVar) 'Restituisce False (intVar non è Null o Empty)
MsgBox VarType (intVar) 'restituisce 2, indicando il tipo Integer
If intVar = 0 Then
MsgBox "Il valore della variabile è zero"
Else
MsgBox "Il valore della variabile non è zero"
End If
End Sub
```

Esempio: Valutare se la variabile è Empty o Null

```
Sub Prova6 ()
Dim var1 As Variant
'restituisce False, var1 non è Null, ma è vuota
MsgBox IsNull (var1)
'la variabile non è inizializzata e restituisce 0, che indica una variabile vuota
MsgBox VarType (var1)
'restituisce il messaggio perché var1 è una variabile vuota
If var1 = 0 And var1 = vbNullString Then
MsgBox "Variabile vuota rappresentata sia come zero (0) che come lunghezza zero (Null) String"
End If
'la variabile viene inizializzata su una stringa di lunghezza zero ("") o vbNullString
var1 = vbNullString
'restituisce False, var1 non è una variabile Null
MsgBox IsNull (var1)
`restituisce 8, che indica una variabile stringa
MsgBox VarType (var1)
'si assegna Null alla variabile
var1 = Null
'restituisce True, una variabile Null, non contenente dati validi
MsgBox IsNull (var1)
'restituisce 1, indicando una variabile Null
MsgBox VarType (var1)
'assegnare dei dati validi alla variabile
var1 = 12
'restituisce False, per una variabile che contiene dati validi
MsgBox IsNull (var1)
'restituisce 2, indicando una variabile integer
MsgBox VarType (var1)
'restituisce False, per un'espressione contenente dati validi
MsgBox IsNull ("Ciao")
End Sub
```

Esempio: Controllare una variabile Null

```
Sub Prova7 ()
'si assegna Null alla variabile
var1 = Null
'restituisce 1, indicando una variabile NUll
MsgBox VarType (var1)
'restituisce il messaggio, indicando una variabile Null
If VarType (var1) = vbNull Then
MsgBox "Variabile Null"
End If
'un'espressione contenente Null restituisce ancora Null
var2 = Null + 2
'restituisce 1, indicando una variabile Null
MsgBox VarType (var2)
End Sub
```

Esempio: Controllare una cella del foglio di lavoro

```
Sub Prova8 ()
Dim var1 As Variant
'restituisce True
MsgBox vbNullString = ""
'se  ActiveCell è vuota restituisce True
MsgBox ActiveCell.Value = ""
MsgBox ActiveCell.Value = vbNullString
MsgBox ActiveCell.Value = 0
MsgBox IsEmpty (ActiveCell.Value)
'assegnare il valore della cella attiva alla variabile
var1 = ActiveCell.Value
'restituisce True
MsgBox IsEmpty (var1)
MsgBox var1 = vbNullString
MsgBox var1 = ""
MsgBox var1 = 0
'restituisce False
MsgBox VarType (var1) = vbNull
'restituisce 0, che indica una variabile vuota
MsgBox VarType (var1)
'se si immette "" nella cella attiva restituisce True
MsgBox ActiveCell.Value = ""
MsgBox ActiveCell.Value = vbNullString
'restituisce False
MsgBox ActiveCell.Value = 0
MsgBox IsEmpty (ActiveCell.Value)
End Sub
```

La Funzione Nothing

L'assegnazione della parola chiave **Nothing** a una variabile oggetto dissocia la variabile stessa da un oggetto reale e questa assegnazione avviene utilizzando l'istruzione Set. Abbiamo visto in precedenza che ogni assegnazione eseguita a delle variabili vengono utilizzate delle risorse di sistema per alloccare in memoria la varibile. Le risorse di sistema e di memoria vengono rilasciati solo dopo aver assegnato Nothing tramite l'istruzione Set a tutte le variabili oggetto che di dissociare queste variabili dall'oggetto reale, o quando tutte le variabili oggetto vengono distrutte. *Si consiglia di impostare esplicitamente tutte le variabili oggetto a Nothing al termine della procedura* o anche prima durante l'esecuzione, quando il codice ha finito di usarle, e questo rilascerà memoria allocata per queste variabili.

Per controllare se un oggetto è stato assegnato o impostato, si utilizza la parola chiave **IsNothing**, vale a dire usando un'espressione come la seguente: *If object_variable Is Nothing*

Esempio: Utilizzare la parola chiave Nothing con una variabile oggetto

```
Sub Prova9()
Dim OVar As Object
'restituisce True, perché non è stato assegnato un oggetto reale alla variabile oggetto
MsgBox OVar Is Nothing
Set OVar = ActiveSheet
'restituisce False, perché è stato assegnato un oggetto reale (foglio) alla variabile
MsgBox OVar Is Nothing
Set OVar = Nothing
'restituisce "La variabile non è associata a un oggetto reale", perché abbiamo dissociato
'la variabile oggetto da un oggetto reale
If OVar Is Nothing Then
MsgBox "La variabile non è associata a un oggetto reale"
Else
MsgBox "Un oggetto reale è assegnato a una variabile Object"
End If
End Sub
```

La funzione Missing

Quando un valore esterno deve essere utilizzato da una procedura per eseguire un'azione, si passa alla procedura da variabile che sono chiamati argomenti. Un argomento è il valore fornito dal codice chiamante a una procedura quando viene chiamato e quando il set di parentesi, dopo il nome della procedura nella dichiarazione Sub o Function, è vuota, si tratta di un caso in cui la procedura non riceve argomenti. Tuttavia, quando gli argomenti sono passati a una procedura da altre procedure, allora questi sono elencati o dichiarati tra le parentesi.

Gli argomenti possono essere specificati come facoltativi, utilizzando la parola chiave *Optional* prima dell'argomento alla sua sinistra e quando si specifica un argomento come opzionale, tutti gli altri argomenti successivi posti alla destra dell'argomento sono specificati come Optional. Si noti che specificando la parola chiave Optional rende un argomento opzionale altrimenti sarà richiesto l'argomento.

L'argomento opzionale dovrebbe essere (anche se non è necessario) dichiarato come tipo di dati Variant per consentire l'uso della funzione **IsMissing** che funziona solo quando viene utilizzato con le variabili dichiarate come Variant. La funzione IsMissing viene utilizzata per determinare se l'argomento opzionale è stato passato alla procedura o meno in modo che ci si può regolare di conseguenza nel codice senza restituire un errore.

Se l'argomento opzionale non è dichiarato come Variant, la funzione IsMissing non funziona, e all'argomento opzionale verrà assegnato il valore predefinito per il tipo di dati che è 0 per le variabili di tipo numerico (cioè Integer, Double, ecc) e Nothing (un riferimento nullo) per le variabili String o variabili di tipo Object.

La funzione IsMissing viene utilizzata con questa sintassi: *IsMissing (argname)* e restituisce un valore booleano, True se non viene passato nessun valore per l'argomento opzionale, e False se un valore è stato passato. Se la funzione IsMissing restituisce True per un argomento, utilizzando l'argomento mancante nel codice causerà un errore, e quindi utilizzando questa funzione aiuterà a regolare il codice di conseguenza.

Esempio: Utilizzo della funzione IsMissing per verificare se un argomento è mancante

```
Function NomeC(Pnome As String, Optional Snome As Variant) As String
'La dichiarazione della procedura contiene due argomenti, il secondo argomento è
'specificato come Optional. 'Dichiarare l'argomento opzionale come tipo di dati Variant
'consentirà l'utilizzo della funzione IsMissing.
If IsMissing(Snome) Then
NomeC = Pnome
Else
NomeC = Pnome & "" & Snome
End If
End Function

 Sub Pas_Nome()
 Dim nome1 As String
 nome1 = InputBox("Inserire il nome")
 'Specificando solo il primo argomento e omettere il secondo argomento che è facoltativo
MsgBox NomeC(nome1)
End Sub
```

Metodi e Proprietà del Foglio di lavoro o Worksheet

L'oggetto ***foglio*** rappresenta un singolo foglio in una cartella e si riferisce a una raccolta di tutti i fogli in una cartella di lavoro, cioè tutti i fogli di lavoro, fogli grafici, macro etc. ed è un membro sia della collezione Worksheets (oggetto foglio di lavoro) che della raccolta Sheets (oggetto Foglio). La proprietà ***Workbook.Worksheets*** restituisce un insieme Worksheets (cioè un oggetto foglio), che si riferisce a tutti i fogli di una cartella di lavoro, mentre la proprietà *Workbook.Sheets* restituisce un insieme Sheets, che si riferisce a tutti i fogli di una cartella di lavoro. Utilizzando il codice *MsgBox ActiveWorkbook.Worksheets.Count* restituirà il numero dei fogli di lavoro nella cartella di lavoro attiva, e il codice *MsgBox ActiveWorkbook.Sheets.Count* restituirà il numero dei fogli nella cartella di lavoro attiva.

L'oggetto *Charts* (grafico) rappresenta un diagramma in una cartella di lavoro, che può essere sia un grafico incorporato o un foglio grafico separato e la collezione Charts si riferisce ad una raccolta di tutti i fogli grafici in una cartella, ed esclude tutti i grafici incorporati, mentre invece l'oggetto *ChartObject* rappresenta un grafico incorporato e si riferisce ad una raccolta di tutti gli oggetti ChartObject in un unico foglio, cioè in un foglio grafico specifico.

Una cartella di lavoro può contenere quattro tipi di fogli, il foglio di lavoro, un foglio con grafico, fogli macro (Macro MS Excel 4.0) e un foglio di dialogo (finestra di dialogo MS Excel 5.0). I fogli macro (chiamati anche macro XLM) e i fogli di dialogo (utilizzati nelle versioni precedenti di Excel per creare finestre di dialogo personalizzate, ora sostituiti dalle UserForm), sono ancora forniti e supportati in Excel 2007 solo per compatibilità con le versioni precedenti di Microsoft Excel, mentre un foglio macro (o un foglio di dialogo) non è incluso come parte della collezione dei fogli di lavoro, ma è una parte della collezione Sheets.

Esempio: Illustrare i tipi di oggetti spiegati sopra, considerando una cartella di lavoro con tre fogli di lavoro denominati: Foglio1, Foglio2 e Foglio3), 2 grafici incorporati in Foglio1, 1 foglio grafico Chart1.

```
Sub prova1()
Dim ws As Worksheet, i As Integer
MsgBox ThisWorkbook.Sheets.Count 'Restituisce 6, in quanto ci sono: 3 fogli
For i = 1 To ThisWorkbook.Sheets.Count 'Restituisce i nomi di ognuno dei 6 fogli
MsgBox ThisWorkbook.Sheets(i).Name
Next i
MsgBox ThisWorkbook.Worksheets.Count 'Restituisce 3 (che sono i fogli di lavoro)
For Each ws In ThisWorkbook.Worksheets 'Restituisce i nomi di ciascuno dei 3 fogli di lavoro
MsgBox ws.Name
Next
Dim sh As Object
For Each sh In ThisWorkbook.Sheets 'Restituisce i nomi di ognuno dei 6 fogli
MsgBox sh.Name
Next
For Each sh In ThisWorkbook.Worksheets 'Restituisce i nomi di ognuno dei 3 fogli di lavoro
MsgBox sh.Name
Next
MsgBox ThisWorkbook.Charts.Count 'restituisce 1 - c'è 1 oggetto grafico "Chart1"
MsgBox Sheets("Chart1").ChartObjects.Count 'restituisce 0 - non c'è grafico nel foglio
'restituisce 2 - ci sono 2 grafici incorporati ChartObject nel foglio di lavoro "Foglio1"
MsgBox Sheets("Sheet1").ChartObjects.Count
End Sub
```

Riferimenti a un singolo foglio di lavoro

Per fare riferimento o restituire un oggetto foglio di lavoro (unico foglio di lavoro), usiamo le proprietà di entrambi i fogli di lavoro e l'oggetto foglio di lavoro, come illustrato di seguito. La proprietà Item dell'oggetto foglio; es. *Worksheets.Item* si riferisce ad un singolo foglio in una collezione e presenta la seguente sintassi: *WorksheetsObject.Item (Index)*, dove *Index* è il nome del foglio di lavoro o il numero di indice, che però è anche possibile omettere utilizzando il 'punto' in una sintassi come:

```
WorksheetsObject (WorksheetName)
'oppure
WorksheetsObject (IndexNumber)
```

Il numero di indice inizia da 1, viene visualizzato per primo ed è collocato all'estrema sinistra del foglio di lavoro nella barra delle schede della cartella di lavoro che si incrementa verso destra per ogni foglio presente, inclusi i fogli di lavoro nascosti, e l'ultimo foglio collocato all'estrema destra del foglio di lavoro viene restituito da *Worksheets (Worksheets.Count)* . La proprietà *Count* dell'oggetto fogli di lavoro restituisce il numero di fogli nella collezione (cioè nella cartella di lavoro), inoltre per riferirsi a un singolo foglio nella raccolta fogli è possibile utilizzare la proprietà ***Sheets.Item*** con la seguente sintassi:*SheetsObject.Item (Index)*

La scheda del foglio di lavoro visualizza il nome del foglio stesso e si utilizza la proprietà ***Name*** dell'oggetto Worksheet per impostare o restituire il nome di un foglio di lavoro usando questa sintassi: *WorksheetObject.Name*. Di seguito alcuni esempi di codice con riferimento a un foglio di lavoro

Restituisce il nome del primo foglio nella cartella attiva, sia usando o omettendo l'istruzione "Item"

```
MsgBox ActiveWorkbook.Worksheets.Item (1). Name
'oppure
MsgBox ActiveWorkbook.Worksheets (1). Name
```

Restituisce il nome del primo foglio della cartella di attiva, sia usando o omettendo l'istruzione "Item"

```
MsgBox ActiveWorkbook.Sheets.Item (1). Name
'oppure
MsgBox ActiveWorkbook.Sheets (1). Name
```

Restituisce il nome del primo foglio ThisWorkbook, omettendo l'istruzione "Item"

```
MsgBox ThisWorkbook.Worksheets (1). Name
```

Attiva il foglio di lavoro denominato Foglio3 della cartella di lavoro attiva

```
ActiveWorkbook.Worksheets ("Foglio3"). Activate
```

Attiva il foglio denominato Chart1 della cartella di lavoro attiva

```
ActiveWorkbook.Sheets ("Chart1"). Activate
```

Non specificando una cartella di lavoro si farà riferimento alla cartella di lavoro attiva, si usa il seguente codice per restituire il nome del secondo foglio di lavoro nella cartella di lavoro attiva:

```
MsgBox Worksheets (2). Name
'Oppure
MsgBox Sheets(2). Name
```

Consultare il foglio attivo nella cartella di lavoro attivo

```
MsgBox ActiveSheet.Name
```

Restituisce il nome del secondo foglio di lavoro nella cartella di lavoro prova.xlsx

```
MsgBox Workbooks ("prova.xlsx"). Worksheets (2). Name
```

Restituisce il nome del primo foglio di lavoro nella seconda cartella di lavoro

```
MsgBox Workbooks(2).Worksheets(1).Name
MsgBox Workbooks.Item(2).Worksheets.Item(1).Name
MsgBox Workbooks(2).Worksheets.Item(1).Name
```

Esempio: Consultare i fogli di lavoro della cartella di lavoro attiva contenente 3 fogli, vale a dire Foglio1, Foglio2 e Foglio3

```
Sub prova2()
'Restituisce il nome del primo foglio di lavoro "Foglio1"
MsgBox ActiveWorkbook.Worksheets(1).Name
'Cambia il nome del primo foglio di lavoro da "Foglio1" a "pippo"
ActiveWorkbook.Worksheets(1).Name = "pippo"
'Restituisce il nuovo nome del primo foglio di lavoro (pippo)
MsgBox Worksheets(1).Name
'Restituisce il nome dell'ultimo foglio (Foglio3)
MsgBox ActiveWorkbook.Worksheets(Worksheets.Count).Name
'Restituisce 3, il n° totale dei fogli di lavoro
MsgBox ActiveWorkbook.Worksheets.Count
'Restituisce il valore -1, che indica che il foglio di lavoro "Foglio2" è visibile
MsgBox Worksheets("Sheet2").Visible
'Nasconde "Foglio2”
Worksheets("Sheet2").Visible = False
'Restituisce il valore 0, che indica che "Foglio2" è nascosto
MsgBox Worksheets("Sheet2").Visible
'Restituisce 3, il n° totale di fogli di lavoro contando anche quelli nascosti
MsgBox Worksheets.Count
'Rende visibile "Foglio2"
Worksheets("Sheet2").Visible = True
End Sub
```

Facendo riferimento a un intervallo di celle, omettendo il qualificatore oggetto – *worksheet object* - per impostazione predefinita assume *Active Sheet*, vale a dire che utilizzando il codice *Range ("A1")* restituirà la cella A1 del foglio attivo, e sarà lo stesso che utilizzare *Application.Range ("A1")* o*ActiveSheet.Range ("A1")*.

Esempio: Immettere il valore 10 nella cella A1 del foglio di lavoro attivo

```
ActiveSheet.Range ("A1"). Value = 10
Range ("A1"). Value = 10
Application.Range ("A1"). Value = 10
```

L'oggetto attivo o ActiveSheet

Se non viene specificata la cartella di lavoro, Excel si riferisce alla cartella di lavoro corrente o attiva di default e nel codice VBA è possibile anche consultare l'attuale cartella di lavoro attiva come *ActiveWorkbook* o ***ActiveSheet*** e entrambe le espressioni *Worksheets (1). Name* e *ActiveWorkbook.Worksheets (1). Name* restituiranno il nome del primo foglio di lavoro nella cartella di lavoro attiva che diventa anche l'oggetto di default in questo caso. Allo stesso modo, entrambe le espressioni *Range ("A1"). Value = 56* e *ActiveSheet.Range ("A1"). Value = 56* inseriranno il valore 56 nella cella A1 del foglio di lavoro attivo nella cartella di lavoro attiva. Questa è una regola generale che omettendo il riferimento di una cartella di lavoro o foglio di lavoro ci si riferisce alla cartella di lavoro attiva o di default, ma è soggetto alle seguenti condizioni:

- Quando il codice VBA viene inserito nei moduli foglio, omettere il riferimento ad un foglio di lavoro si farà riferimento al foglio specifico in cui il il codice viene inserito nel modulo e non al *foglio attivo*
- Quando il codice VBA viene inserito nel modulo *Workbook (ThisWorkbook)*, omettere il riferimento ad una cartella di lavoro si farà riferimento alla cartella di lavoro in cui è inserito il codice e **NON** alla *cartella di lavoro attiva*. Questo significa che, omettendo il riferimento ad un foglio di lavoro si imposterà *ActiveSheet* quando il codice VBA viene inserito nei moduli standard di codice (Module1, Module2, etc.) oppure nel modulo della cartella di lavoro (ThisWorkbook) e **NON** quando il codice VBA viene inserito nei moduli Foglio (Foglio1, Foglio2, etc.) o Form o eventuali moduli di classe
- Omettendo il riferimento a una cartella di lavoro, quella predefinita sarà *ActiveWorkbook* quando il codice VBA viene inserito nei moduli di codice standard (Module1, Module2, etc.) o nei moduli foglio (Foglio1, Foglio2, etc.) e **NON** quando il codice VBA viene inserito nel modulo *Workbook*(ThisWorkbook).

Codice Name e Sheet Name

Nel Progetto VBE, la cartella *Objects* è sempre presente e contiene un oggetto foglio per ogni foglio di lavoro esistente, e un oggetto ThisWorkbook. Ogni oggetto foglio ha come primo nome il nome in codice del foglio, che appare al di fuori delle parentesi, e come secondo nome, che compare dopo il nome in codice e tra parentesi, il nome della scheda del foglio che appare nel foglio di lavoro di Excel. Il nome in codice del foglio di lavoro selezionato viene visualizzato a destra del "nome" nella finestra Proprietà, mentre il nome del foglio viene visualizzata a destra del nome quando si scorre verso il basso nella finestra Proprietà. Per impostazione predefinita, quando si aggiunge un foglio di lavoro, il nome in codice e il nome del foglio sono gli stessi e i nomi in codice di default e i fogli di default iniziano da Foglio1, Foglio2 etc. in questo ordine da sinistra a destra.

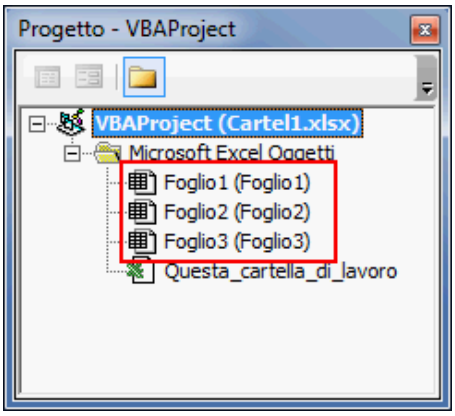


Fig. 1

È possibile modificare sia il nome in codice che il nome del foglio, mentre il nome del foglio può essere modificato nella finestra Proprietà (in VBE) o nella scheda foglio o da codice VBA, ma il nome in codice può essere modificato solo nella finestra Proprietà e non a livello di programmazione. Sia il nome in codice che il nome del foglio possono essere utilizzati durante la scrittura del codice. Mostriamo di seguito come utilizzare questi nomi in codice VBA.

Il codice seguente inserisce il testo "ciao" nella cella A1 del foglio di lavoro il cui nome in codice è Foglio1

```
Sheet1.Range ("A1"). Value = "ciao"
```

Il codice seguente inserisce il testo "ciao" nella cella A1 del foglio di lavoro il cui nome foglio è Foglio1

```
Worksheets ("Foglio1"). Range ("A1"). Value = "ciao"
```

Il codice seguente inserisce il testo "ciao" nella cella A1 del primo foglio di lavoro

```
Worksheets (1). Range ("A1"). Value = "ciao"
```

Attivare o selezionare un foglio di lavoro

Il foglio di lavoro attivo è il foglio di lavoro che si sta attualmente visualizzando o lavorare e la proprietà *Workbook.ActiveSheet* restituisce il foglio attualmente attivo in una cartella di lavoro: Sintassi: *WorkbookObject.ActiveSheet*. Per rendere un foglio attivo si deve utilizzare il metodo *Worksheet.Activate* con questa sintassi: ***WorksheetObject.Activate***. Si noti che è possibile selezionare più fogli di lavoro, ma si può attivare solo un singolo foglio. Per selezionare un foglio di lavoro si utilizza il metodo *Worksheet.Select* con questa sintassi: *WorksheetObject.Select (Replace)*. *Replace* è un argomento facoltativo, usato solo quando si utilizza il metodo Select per i fogli (il metodo Select è utilizzato anche per selezionare le celle) e ponendolo a True, questo argomento sostituirà la selezione precedente, mentre con False includerà la selezione precedente (estendendo così la selezione corrente per includere i fogli selezionati in precedenza) mentre omettendo la specifica, l'argomento sarà posto di default a True. Si utilizza il metodo *Worksheets.Select* per selezionare tutti i fogli in una cartella di lavoro.
Sintassi: *WorksheetsObject.Select (Replace)*.

Esempio: Selezionare fogli singoli o multipli considerando una cartella di lavoro contenente tre fogli di lavoro

```
Sub seleziona1()
ActiveWorkbook.Worksheets(2).Select 'Seleziona e attiva il secondo foglio di lavoro Foglio2
MsgBox ActiveWorkbook.ActiveSheet.Name 'Restituisce il foglio di lavoro attivo Foglio2
'Selezionare più fogli di lavoro nella cartella utilizzando un array di nomi di foglio
’dove il primo foglio nella matrice è Foglio3 che diventa il foglio di lavoro attivo
ActiveWorkbook.Sheets(Array("Foglio3", "Foglio1")).Select
MsgBox ActiveWorkbook.ActiveSheet.Name 'Restituisce il foglio di lavoro attivo Foglio3
'Seleziona e attivare il secondo foglio di lavoro Foglio2, la selezione precedente
’viene sostituita quindi questo è l'unico foglio di lavoro selezionato
ActiveWorkbook.Worksheets("Foglio2").Select
MsgBox ActiveWorkbook.ActiveSheet.Name 'Restituisce il foglio di lavoro attivo Foglio2
Dim i As Integer 'Selezionare più fogli di lavoro , tranne l'ultimo
For i = 1 To ThisWorkbook.Worksheets.Count - 1
ActiveWorkbook.Worksheets(i).Select (False)
Next i
'Restituisce il foglio di lavoro attivo Foglio2, che era il foglio di lavoro attivo prima del ciclo For
MsgBox ActiveWorkbook.ActiveSheet.Name
End Sub
```

Esempio: Selezionare e attivare i fogli di lavoro, considerando una cartella di lavoro che contenente 4 fogli, vale a dire Foglio1, Foglio2, Foglio3 e Foglio4

```
Sub seleziona2()
ActiveWorkbook.Worksheets(1).Select 'Seleziona e attiva il primo foglio (Foglio1)
MsgBox ActiveWorkbook.ActiveSheet.Name 'Restituisce il foglio di lavoro attivo (Foglio1)
'Seleziona il secondo foglio (Foglio2), senza attivarlo e senza deselezionare la selezione precedente
ActiveWorkbook.Worksheets(2).Select (False)
MsgBox ActiveWorkbook.ActiveSheet.Name 'Restituisce il foglio di lavoro attivo (Foglio1)
'Seleziona il terzo foglio (Foglio3), senza attivarlo e senza deselezionare la selezione precedente
ActiveWorkbook.Worksheets(3).Select (False)
MsgBox ActiveWorkbook.ActiveSheet.Name 'Restituisce il foglio di lavoro attivo (Foglio3)
'Attiva il terzo foglio di lavoro (Foglio3), senza deselezionare la selezione precedente
 `perché uno dei fogli selezionati viene attivato
ActiveWorkbook.Worksheets(3).Activate
MsgBox ActiveWorkbook.ActiveSheet.Name 'Restituisce il foglio di lavoro attivo (Foglio3)
'I fogli di lavoro (Foglio1, Foglio2 e Foglio3) sono attualmente selezionati
Dim ws As Worksheet
For Each ws In ActiveWindow.SelectedSheets
MsgBox ws.Name
Next
'Attiva e seleziona il quarto foglio di lavoro (Foglio4), che è l'unico foglio selezionato ora,
 `deselezionando la selezione precedente, perché il foglio attivo non è uno dei fogli di lavoro
 `selezionati in precedenza
ActiveWorkbook.Worksheets(4).Activate
MsgBox ActiveWorkbook.ActiveSheet.Name 'Restituisce il foglio di lavoro attivo (Foglio4)
End Sub
```

Esempio: Selezionare tutti i fogli, considerando una cartella di lavoro contenente 3 fogli di lavoro, Foglio1, Foglio2 e Foglio3

```
Sub seleziona3()
ActiveWorkbook.Worksheets.Select 'Seleziona tutti i fogli di lavoro nella cartella di lavoro attiva
MsgBox ActiveWorkbook.ActiveSheet.Name 'Il primo foglio diventa il foglio attivo -Foglio1
'Seleziona tutti i fogli della cartella di lavoro attiva utilizzando una matrice di nomi
ActiveWorkbook.Sheets(Array("Foglio2", "Foglio3", "Foglio1")).Select
'Il primo foglio nella matrice diventa il foglio attivo e restituisce Foglio2
MsgBox ActiveWorkbook.ActiveSheet.Name
'attivare e selezionare il terzo foglio di lavoro (Foglio3), deselezionando tutti gli altri fogli
ActiveWorkbook.Worksheets(3).Activate
MsgBox ActiveWorkbook.ActiveSheet.Name 'Restituisce il foglio di lavoro attivo (Foglio3)
'Modo alternativo per selezionare tutti i fogli della cartella di lavoro attiva
Dim ws As Worksheet
For Each ws In ActiveWorkbook.Worksheets
If ws.Visible Then ws.Select (False)
Next
MsgBox ActiveWorkbook.ActiveSheet.Name 'il foglio attivo rimane (Foglio3)
End Sub
```

La proprietà Window.SelectedSheets

Si utilizzare la proprietà *Window.SelectedSheets* per determinare tutti i fogli selezionati in una finestra specificata con la seguente sintassi:*WindowObject.SelectedSheets*, dove Window (1) si riferisce sempre alla finestra attiva. Una sola finestra (Window Object) è un membro della Collezione di Windows e la raccolta di Windows contiene tutte le finestre dell'Applicazione Excel (raccolta di Windows per l'oggetto Application) o contiene tutte le finestre della cartella di lavoro specificata (raccolta di Windows per oggetto Workbook).

Aggiungere e rinominare Fogli di lavoro

Per creare o aggiungere un nuovo foglio di lavoro in una cartella di lavoro specificato, si utilizzare il metodo *Sheets.Add*, che diventa anche il foglio attivo con questa sintassi: *SheetsObject.Add(Before, After, Count, Type)* . I nomi di default dei fogli sono Foglio1, Foglio2 etc. in ordine da sinistra a destra e tutti gli argomenti sono opzionali. Si utilizza l'argomento *Before* per specificare il foglio prima del quale si desidera aggiungere il nuovo foglio, mentre si utilizza l'argomento *After* per specificare il foglio dopo il quale si desidera aggiungere il nuovo foglio. L'argomento *Count* specifica il numero di fogli che si desidera aggiungere, di default è uno, mentre si utilizza l'argomento *Type* per specificare un valore costante o (costante XlSheetType) che indica il tipo di foglio da aggiungere che possono essere: XlSheetType, xlWorksheet, xlChart, xlExcel4MacroSheet, xlExcel4IntlMacroSheet o xlDialogSheet. Esempi di aggiunta di un foglio

Utilizzando il metodo ***Add*** senza specificare alcun argomento, si aggiunge un nuovo foglio di lavoro prima del foglio attivo e se gli argomenti Before e After vengono omessi il valore predefinito dell'argomento Count è 1

```
ActiveWorkbook.Worksheets.Add
```

Utilizzando il metodo **Add** e specificando l'argomento **After**, si aggiunge un nuovo foglio dopo il foglio di lavoro denominato Foglio2

```
ActiveWorkbook.Worksheets.Add After: = Worksheets ("Foglio2")
```

Utilizzando il metodo Add e specificando i due argomenti After e **Count**, si aggiungono 3 nuovi fogli di lavoro dopo il foglio di lavoro Foglio2

```
ActiveWorkbook.Worksheets.Add After: = Worksheets ("Foglio2"), Count: = 3
```

Utilizzando il metodo Add e specificando i due argomenti After e Count, si aggiungono 2 nuovi fogli di lavoro dopo il foglio di lavoro denominato Foglio2

```
ActiveWorkbook.Worksheets.Add, Worksheets ("Foglio2"), 2
```

Nomi di fogli predefiniti

Abbiamo visto poco sopra come creare una nuova cartella di lavoro utilizzando il metodo *Workbooks.Add* (riferito all'oggetto Workbooks) e a meno che non si specifichi un file Excel come modello per la nuova cartella di lavoro, il metodo Add creerà una nuova cartella di lavoro di Excel con il default di tre fogli bianchi, in cui il numero predefinito di fogli può essere modificato utilizzando la proprietà *Application.SheetsInNewWorkbook*. I tre fogli bianchi avranno di default i nomi Foglio1, Foglio2 e Foglio3, in ordine da sinistra a destra.

Si utilizza la proprietà ***Name*** dell'oggetto Worksheet (Worksheet.Name) per impostare o restituire il nome di un foglio di lavoro Sintassi:*WorksheetObject.Name*. Alcuni esempi di utilizzo della proprietà Nome:
Restituisce il nome del primo foglio di lavoro nella cartella di lavoro attiva

```
MsgBox ActiveWorkbook.Worksheets (1). Name
```

Attivare il foglio di lavoro denominato Foglio1 della cartella di lavoro attiva

```
ActiveWorkbook.Worksheets ("Foglio1"). Activate
```

Modificare il nome del foglio di lavoro denominato Foglio1 nella cartella di lavoro attiva prova1

```
ActiveWorkbook.Worksheets ("Foglio1"). Name = "prova1"
```

Utilizzando il metodo Add e specificando l'argomento Before si aggiunge un nuovo foglio di lavoro prima del foglio di lavoro Foglio2, e utilizzando la proprietà Name, si rinomina il nuovo foglio di lavoro pippo

```
ActiveWorkbook.Worksheets.Add (Before: = Worksheets ("Foglio2")). Name = "pippo"
```

Restituire i nomi di tutti i fogli (fogli di lavoro, grafici, fogli macro e fogli di dialogo) in ThisWorkbook

```
Dim i As Integer
For i = 1 To ThisWorkbook.Sheets.count
MsgBox ThisWorkbook.Sheets (i). Name
Next i
```

Restituire i nomi di tutti i fogli di lavoro (esclusi grafici, macro e fogli di dialogo) nella cartella di lavoro attiva

```
Dim ws As Worksheet
For Each ws In ActiveWorkbook.Worksheets
MsgBox ws.Name
Next
```

Esempio: Aggiungere un nuovo foglio, e consultarlo utilizzando la sua variabile oggetto. Per utilizzare la variabile oggetto attraverso la procedura, verrà dichiarata pubblica (Public), nella prima linea del modulo.

```
Sub foglio1()
Dim wsNew As Worksheet
Set wsNew = Worksheets.Add
wsNew.Name = "Pippo1"
'Riferimento al nuovo foglio - il nuovo foglio diventa il foglio attivo
Worksheets.Add
ActiveSheet.Name = "Pippo2"
End Sub
```

Copiare un foglio di lavoro

Si utilizza il metodo ***Worksheet.Copy*** per copiare un foglio di lavoro e posizionarlo, prima o dopo un altro foglio in una cartella di lavoro, o per creare una nuova cartella di lavoro contenente il foglio copiato utilizzando il metodo *Sheets.Copy* per copiare un foglio usando la seguente sintassi: *WorksheetObject.Copy (Before, After)*. Entrambi gli argomenti *Before* e *After* sono opzionali, ed è possibile specificare solo uno alla volta di questi e questi argomenti fanno riferimento al foglio *prima* o *dopo* dove il foglio copiato sarà posto, e omettendo entrambi gli argomenti si creerà una nuova cartella di lavoro contenente il foglio copiato. È possibile copiare un foglio di lavoro in una posizione all'interno della stessa cartella di lavoro o in un'altra cartella di lavoro.

Copiare il foglio denominato Foglio1 e posizionarlo nella cartella attiva prima di Foglio3

```
Worksheets ("Foglio1"). Copy Before: = Sheets ("Foglio3")
```

Copiare il foglio denominato Foglio1 e posizionarlo nella cartella attiva dopo il Foglio3

```
Worksheets ("Foglio1"). Copy After: = Sheets ("Foglio3")
```

Copiare Foglio1 dalla cartella di lavoro attiva e metterlo prima del foglio pippo nella cartella prova.xlsm

```
Worksheets ("Foglio1"). Copy Before: = Workbooks ("prova.xlsm"). Sheets ("pippo")
```

Copiare Foglio1 dalla cartella attiva in una nuova cartella di lavoro

```
Worksheets ("Foglio1"). Copy
```

Spostare o Cambiare Sequenza al foglio

Si utilizza il metodo ***Worksheet.Move*** per spostare un foglio di lavoro e posizionarlo, prima o dopo un altro foglio in una cartella di lavoro, o per creare una nuova cartella di lavoro contenente il foglio spostato (si utilizza il metodo *Sheets.Move* per spostare un foglio) con questa sintassi: *WorksheetObject.Move (Before, After)*. Entrambi gli argomenti *Before* e *After* sono opzionali, ed è possibile specificare solo uno di questi alla volta, inoltre questi argomenti fanno riferimento al foglio *prima* o *dopo* dove verrà inserito il foglio spostato, e omettendo entrambi gli argomenti si creerà una nuova cartella di lavoro contenente il foglio spostato. È possibile spostare un foglio di lavoro in una posizione all'interno della stessa cartella di lavoro o ad un'altra cartella di lavoro e lo spostamento di un foglio di lavoro modifica la sequenza dei fogli di lavoro, nelle schede che appaiono in una cartella di lavoro, ma non in Esplora progetti in VBE.

Spostare il foglio denominato Foglio1 nella cartella attiva e posizionarlo prima del Foglio3

```
Worksheets ("Foglio1"). Move Before: = Sheets ("Foglio3")
```

Spostare il foglio denominato "Foglio1" nella cartella attiva e posizionarlo dopo il "Foglio3"

```
Worksheets ("Foglio1"). Move After: = Sheets ("Foglio3")
```

Spostare il Foglio1 dalla cartella di lavoro attiva e metterlo prima del foglio pippo nella cartella prova.xlsm

```
Worksheets ("Foglio1"). Move Before: = Workbooks ("prova.xlsm"). Sheets ("pippo")
```

Spostare il Foglio1 dalla cartella attiva, e creare una nuova cartella di lavoro che contiene il foglio spostato

```
Worksheets ("Foglio1"). Move
```

Nascondere o rendere visibile un foglio di lavoro

Per nascondere un foglio di lavoro o per renderlo visibile si utilizzare la proprietà *Worksheet.Visible* (*Sheets.Visible*), utilizzando semplicemente il codice*Worksheets ("Foglio1"). Visible = False* per nascondere e *Worksheets ("Foglio1"). Visible = True* per renderlo visibile. L' enumerazione

- ***XlSheetVisibility*** viene utilizzata per impostare o restituire un valore che indica se il foglio è visibile o nascosto
- ***xlSheetHidden*** si nasconde un foglio di lavoro
- ***xlSheetVeryHidden*** si nasconde un foglio di lavoro che può essere reso visibile solo attraverso codice VBA
- ***xlSheetVisible*** si renderà visibile il foglio.

Esempio: Utilizzare i metodi per nascondere, rendere visibile o rendere molto nascosto il foglio1

```
Sub nascondi1()
Worksheets("Foglio1").Visible = False 'Nasconde Foglio1
'Restituisce il valore 0, che indica che Foglio1 è nascosto
MsgBox Worksheets("Foglio1").Visible
Worksheets("Foglio1").Visible = True 'Rende visibile Foglio1
'Restituisce il valore -1, che indica che Foglio1 è visibile
MsgBox Worksheets("Foglio1").Visible
Worksheets("Foglio1").Visible = 2 'Rende Foglio1 molto nascosto
'Restituisce il valore 2, Foglio1 è molto nascosto
MsgBox Worksheets("Foglio1").Visible
Worksheets("Foglio1").Visible = xlSheetVisible 'Rende visibile Foglio1
'Restituisce il valore -1, che indica che Foglio1 è visibile
MsgBox Worksheets("Foglio1").Visible
End Sub
```

Esempio: Nascondere tutti i fogli tranne l'ultimo, tenendo presente che tutti i fogli in una cartella di lavoro non possono essere nascosti

```
Sub nascondi2()
   Dim oSh As Object, i As Long
Sheets(Sheets.Count).Visible = True
For i = 1 To Sheets.Count - 1
Sheets(i).Visible = xlSheetHidden
Next i
For Each oSh In Sheets
oSh.Visible = xlSheetVisible
Next
End Sub
```

Rimuovere o eliminare fogli

Si utilizzare il metodo ***Worksheet.Delete*** per eliminare o rimuovere un foglio di lavoro in una cartella di lavoro (*Sheets.Delete*) con la seguente sintassi:*WorksheetObject.Delete* e usando questo metodo verrà visualizzata una finestra di dialogo che chiede all'utente di confermare o annullare l'eliminazione. Con l'impostazione della proprietà *Application.DisplayAlerts* a False non verrà visualizzata nessuna richiesta o avviso e in questo caso una risposta predefinita sarà scelta da Excel. Questa proprietà viene ripristinata al valore predefinito (True) dopo che la procedura è termina. Utilizzare il codice*Application.DisplayAlerts = False* per eliminare un foglio di lavoro senza visualizzare nessun avviso

Cancellare il Foglio1 nella cartella di lavoro attiva, senza visualizzare nessun avviso

```
Application.DisplayAlerts = False
Sheets ("Foglio1"). Delete
```

Fogli di lavoro e Layout di pagina

Vediamo ora brevemente alcune opzioni per il layout di pagina di un foglio di lavoro, e come personalizzare le viste della cartelle di lavoro. Per le impostazioni di pagina di un foglio di lavoro si utilizza la proprietà *Worksheet.PageSetup* che si occupa degli attributi della pagina per il foglio di lavoro come l'orientamento, i margini, le dimensioni della carta, e così via. Usando la proprietà PageSetup restituisce un oggetto *PageSetup* e gli attributi sono impostati come proprietà dell'oggetto cioè Orientaton, PrintArea, LeftMargin, RightMargin, etc che sono le proprietà dell'oggetto PageSetup. Vedere il seguente esempio sull'utilizzo di queste proprietà per impostare gli attributi della pagina.

```
Sub setup_pagina()
`impostare gli attributi e poi stampare il foglio
With Worksheets("Foglio3")
 `impostare le proprietà dell'oggetto PageSetup
With .PageSetup
.PrintTitleRows = Rows(1).Address
.Orientation = xlLandscape
.Zoom = 90
.PrintArea = "$D$1:$R$50"
'inserire il numero di pagina nell'intestazione e alllineato al centro
.CenterHeader = "&P"
'Imposta il primo numero di pagina da utilizzare durante la stampa del foglio di lavoro
.FirstPageNumber = 2
.PaperSize = xlPaperA4
.LeftMargin = Application.InchesToPoints(0.25)
.RightMargin = Application.InchesToPoints(0.5)
.TopMargin = Application.InchesToPoints(1)
.BottomMargin = Application.InchesToPoints(0.75)
.HeaderMargin = Application.InchesToPoints(0.5)
.FooterMargin = Application.InchesToPoints(0.25)
'Visualizza la griglia delle celle quando il foglio viene stampato
.PrintGridlines = True
End With
 `stampa il foglio
.PrintOut
End With
End Sub
```

E' possibile visualizzare un'anteprima di come il foglio di lavoro verrà stampato utilizzando il metodo *Worksheet.PrintPreview* che presenta questa sintassi:*WorksheetObject.PrintPreview (EnableChanges)*. E' facoltativo specificare l'argomento *EnableChanges*, che accetta un valore booleano (il valore predefinito è True), per consentire o non consentire all'utente di modificare le opzioni di impostazione della pagina (es. orientamento della pagina, il ridimensionamento, i margini, etc) disponibili in anteprima di stampa. È inoltre possibile applicare il metodo *PrintPreview* a un oggetto cartella di lavoro o a un oggetto Range, per visualizzare un'anteprima di come verrà eseguita la stampa. Esempio – PrintPreview

```
Sub anteprima()
'Visualizzare un'anteprima di stampa del foglio attivo della cartella di lavoro "prova.xlsm,
 `non consentendo all'utente di cambiare le opzioni di pagina disponibili in anteprima di stampa
Workbooks("prova.xlsm").PrintPreview EnableChanges:=False
'Visualizzare un'anteprima di stampa del Foglio3 della cartella "prova.xlsm, permettendo
 `all'utente di cambiare le opzioni di pagina disponibili in anteprima di stampa.
Workbooks("prova.xlsm").Worksheets("Foglio3").PrintPreview EnableChanges:=True
End Sub
```

Inoltre utilizzando la proprietà *Worksheet.DisplayPageBreaks* è possibile visualizzare sia le interruzioni di pagina automatiche che manuali su un foglio di lavoro.Questa è un'impostazione booleana che può essere impostata solo se è installata una stampante e con il codice *ActiveSheet.DisplayPageBreaks = True* verranno visualizzate le interruzioni di pagina sul foglio attivo. Si utilizza la proprietà *Window.View*, per impostare o restituire una cartella di lavoro (foglio di lavoro attivo) come verrà mostrata nella finestra, utilizza questa sintassi: *WindowObject.View*. Si possono avere tre impostazioni per questa struttura:

- ***xlNormalView*** *(valore 1)*
- ***xlPageBreakPreview*** *(valore 2)*
- ***xlPageLayoutView*** *(valore 3)*

Che rispettivamente sono le visualizzazioni della cartella di lavoro definite 'Normale', 'Layout di pagina' e 'interruzione di pagina'. Vedi sotto esempio

```
Sub vista1()
Worksheets("Foglio3").Activate 'Attivare il foglio di lavoro
With ActiveSheet
.Rows(9).PageBreak = xlPageBreakManual `si imposta la posizione di un'interruzione manuale
.Columns("G").PageBreak = xlPageBreakManual
With .PageSetup
.Orientation = xlPortrait
.Zoom = 100
.PrintArea = "$A$1:$L$17"
.CenterHeader = "&P" 'Numero di pagina da stampare allineata al centro nell'intestazione
.FirstPageNumber = 1
End With
End With
ActiveWindow.View = xlPageBreakPreview
ActiveSheet.Rows(9).PageBreak = xlPageBreakNone 'Reset alla visualizzazione normale
ActiveSheet.Columns("G").PageBreak = xlPageBreakNone
ActiveSheet.DisplayPageBreaks = False
ActiveSheet.PageSetup.PrintArea = "" 'l'Intero foglio diventa l'area di stampa
ActiveWindow.View = xlNormalView  'Impostare la visualizzazione normale
ActiveWindow.View = xlPageLayoutView 'Impostare la visualizzazione Layout di pagina
End Sub
```

Nascondere la barra della formula

```
Application.DisplayFormulaBar = False
```

Impostare Excel in modalità a schermo intero

```
Application.DisplayFullScreen = True
```

Nascondere le intestazioni del foglio di lavoro

```
ActiveWindow.DisplayHeadings = False
```

Nascondere la visualizzazione della griglia in un foglio di lavoro

```
ActiveWindow.DisplayGridlines = False
```

Blocca riquadri in un foglio di lavoro specificato

```
Range ("C2"). Select
ActiveWindow.FreezePanes = True
```

Spesso è possibile utilizzare i seguenti codici per togliere la barra della formula da tutti i fogli, o all'attivazione del foglio di lavoro, oppure in altri contesti sfruttando svariati eventi e metodi come si vede dagli esempi sotto riportati

```
Private Sub Workbook_Activate ()
Application.DisplayFormulaBar = False
End Sub

Private Sub Workbook_Deactivate ()
Application.DisplayFormulaBar = True
End Sub

Private Sub Workbook_Open ()
Dim ws As Worksheet, wn As Window
For Each ws In ActiveWorkbook.Worksheets
ws.activate
For Each wn In ActiveWorkbook.Windows
wn.DisplayHeadings = False
Next
Next
Sheets("Foglio1").Activate
End Sub
```

Calcolo nei Fogli di lavoro

La proprietà *Application.Calculation* restituisce o imposta la modalità di calcolo di Excel e dispone di 3 impostazioni:

- *xlCalculationAutomatic* - (Default) ricalcolo automatico come i dati vengono immessi nelle celle
- *xlCalculationSemiautomatic* - Ricalcolo automatico ad eccezione delle tabelle di dati
- *xlCalculationManual* - Il calcolo viene fatto solo quando richiesto dall'utente facendo clic su "Calcola ora" o "Calcola foglio" o premendo F9, oppure con codice VBA.

Impostare la modalità di calcolo manuale:

```
Application.Calculation = xlCalculationManual
```

Si applicare il metodo *Calculate* a un oggetto foglio per calcolare un foglio di lavoro specifico in una cartella di lavoro, oppure si può applicare questo metodo a un oggetto *Application* per calcolare tutte le cartelle di lavoro aperte, oppure è possibile applicare questo metodo a un intervallo di celle di un foglio di lavoro.

Metodo Calcola applicabile all'oggetto Application, calcola tutte le cartelle che sono aperte

```
Application.Calculate
Calculate
```

Il metodo Calcola applicabile all'oggetto foglio di lavoro; calcolare un foglio di lavoro denominato Foglio1

```
Application.Worksheets ("Foglio1"). Calculate
Worksheets ("Foglio1"). Calculate
```

Il metodo Calcola applicabile all'intervallo di celle specificato in un foglio di lavoro

```
Worksheets ("Foglio1"). Range ("A5: B6"). Calculate
```

Calcolare l'intera colonna (colonna A) in un foglio di lavoro

```
Worksheets ("Foglio1"). Columns (1). Calculate
```

Calcola celle non contigue nel foglio di lavoro attivo

```
Range ("A5, A6, B7, B20"). Calculate
```

In Excel, la modalità di calcolo predefinita è la Modalità Automatica (*Application.Calculation = xlCalculationAutomatic*), in cui Excel calcola automaticamente ogni cella come si entra nel foglio, quando Excel è in modalità manuale (*Application.Calculation = xlCalculationManual*), il calcolo viene effettuato solo quando richiesto dall'utente, facendo clic su "Calcola Ora" o premendo F9 o cambiando la modalità di calcolo. In modalità automatica, per ciascun nuovo valore immesso da una macro, Excel ricalcolerà tutte le celle interessate dal nuovo valore rallentando notevolmente l'esecuzione del codice VBA, soprattutto nel caso di grandi macro i cui calcoli sono significativi.

Per accelerare una macro e rendere l'esecuzione più veloce ed efficiente, è tipico disattivare il calcolo automatico all'inizio della macro e ricalcolare il foglio di lavoro specifico utilizzando il metodo Calculate all'interno della macro. Il seguente esempio disattiva gli aggiornamenti dello schermo e i calcoli automatici e utilizza il metodo Calculate, durante l'esecuzione del codice VBA

```
Sub Calcola()
'Disattivare Aggiornamenti schermo e calcoli automatici
Application.ScreenUpdating = False
Application.Calculation = xlCalculationManual
'Inserisci qui il tuo codice
'Utilizzare il metodo Calculate per calcolare tutte le cartelle di lavoro aperte
Application.Calculate
'Attivare gli aggiornamenti dello schermo e calcoli automatici
Application.ScreenUpdating = True
Application.Calculation = xlCalculationAutomatic
End Sub
```

A volte si può impostare la proprietà *Application.CalculateBeforeSave* a True per calcolare le cartelle di lavoro prima di salvarle su disco se la struttura di calcolo è stata impostata su manuale specificando *xlCalculationManual*. La proprietà CalculateBeforeSave accetta valori booleani (Ture o False) e non è influenzata dalla modifica della proprietà di calcolo. In questo caso

```
Application.Calculation = xlCalculationManual
Application.CalculateBeforeSave = True
```

Si imposta la proprietà *Worksheet.EnableCalculation* a True, per ricalcolare automaticamente il foglio di lavoro quando questa proprietà è impostata su False, Excel non ricalcola il foglio, e quando l'impostazione viene modificata da False a True, Excel esegue il ricalcolo. Si noti che questa proprietà non viene mantenuta quando la cartella di lavoro viene chiusa, e tornerà al suo valore predefinito (True) per l'apertura della cartella di lavoro e, quindi, sarà necessario resettare con il codice VBA ogni volta che si apre la cartella di lavoro. Per controllare i calcoli per ogni foglio di lavoro, è possibile impostare la proprietà *EnableCalculation* a False per i fogli di lavoro che non si desidera calcolare, e quindi impostare la proprietà *Application.Calculation* a*xlCalculationAutomatic*. Si noti che l'esecuzione del metodo Calculate su un foglio di lavoro per il quale la proprietà EnableCalculation è impostata su False, non calcolerà tale foglio di lavoro. Il seguente codice calcola automaticamente tutti i fogli tranne Foglio1

```
Worksheets ("Foglio1"). EnableCalculation = False
Application.Calculation = xlCalculationAutomatic
```

Metodi e Proprietà della Cartella di Lavoro

L'oggetto **WorkBook** è il "*figlio*" dell'oggetto Application nella gerarchia degli oggetti di VBA, e rappresenta una singola cartella di lavoro di Excel all'interno dell'applicazione e si riferisce ad un insieme di tutte le cartelle di lavoro attualmente aperte in Excel. Si noti che mentre si lavora con cartelle di lavoro, si utilizzeranno Proprietà e Metodi dell'oggetto cartella di lavoro e per fare riferimento o restituire un oggetto Workbook (singola cartella di lavoro), possono essere utilizzate le seguenti proprietà.

Proprietà Workbooks.Item

L' **Item** dell'oggetto Workbooks si riferisce ad una singola cartella di lavoro in una collezione e viene rappresentata con questa sintassi: *WorkbooksObject.Item (Index)*, dove *Index* è il nome della cartella di lavoro o il numero di indice che sarebbe anche possibile omettere utilizzando una sintassi come:*WorkbooksObject (WorkbookName)* o *WorkbooksObject (IndexNumber)*. Il numero di indice inizia da 1 per il primo foglio aperto o creato e si incrementa per ogni successiva cartella di lavoro, incluse quelle nascoste. L'ultima cartella di lavoro viene restituita dalla proprietà Count in questo modo: *Workbooks.Count*. La Proprietà *WorkbooksObject.Count*, restituisce il numero di cartelle di lavoro. Consultare gli esempi sotto riportati di utilizzo di questa proprietà.

Esempio: Chiudere la cartella di lavoro denominata "VBA_1.xlsm"

```
Workbooks.Item ("VBA_1.xlsm"). Close
'Oppure
Workbooks ("VBA_1.xlsm"). Close
```

Esempio: Chiudere la cartella di lavoro aperta dopo averla salvata

```
Workbooks(Workbooks.Count).Close  SaveChanges:=True
```

Esempio: Restituisce il nome della cartella di lavoro con indice 2

```
MsgBox Workbooks(2).Name
```

Esempio: Restituisce il nome dell'ultima cartella di lavoro aperta o creata

```
MsgBox Workbooks(Workbooks.Count).Name
```

Esempio: Restituire i nomi di tutte le cartelle di lavoro aperte:

```
Sub workbookNames()
Dim i As Integer
For i = 1 To Workbooks.Count
msgbox Workbooks(i).Name
Next i
End Sub
```

Esempio: Impostare una variabile in una cartella di lavoro:

```
Sub workbookVariable()
Dim wb As Workbook, i As Integer
Set wb = Workbooks("Excel_1.xlsx")
For i = 1 To wb.Worksheets.Count
MsgBox wb.Worksheets(i).Name
Next i
End Sub
```

Proprietà ActiveWorkbook

Questa proprietà restituisce la cartella di lavoro attiva, cioè la cartella di lavoro nella finestra attiva con questa sintassi: *ApplicationObject.ActiveWorkbook*, esempio:

```
MsgBox "il nome della cartella attiva è" & ActiveWorkbook.Name
```

Proprietà ThisWorkbook

Questa proprietà viene utilizzata solo dall'interno dell'applicazione Excel e restituisce la cartella di lavoro in cui il codice viene eseguito al momento. Sintassi: *ApplicationObject.ThisWorkbook*, esempio:

```
MsgBox "Il nome di questa cartella di lavoro è" & ThisWorkbook.Name
```

Si noti che sebbene il più delle volte ActiveWorkbook è la stessa ThisWorkbook, ma potrebbe non essere sempre così, la cartella di lavoro attiva può essere diversa da quella in cui viene eseguito il codice, come illustrato dal seguente esempio di codice.

Esempio: Aprire due file della cartella di lavoro di Excel ("prova1.xlsm" e "prova2.xlsm") in un'unica istanza (questo consentirà a tutte le cartelle di lavoro di accedere alla macro), Inserire il codice nella cartella di lavoro "prova1.xlsm", che è anche la cartella di lavoro attiva

```
Sub ActiveWorkbook_ThisWorkbook()
MsgBox "Il nome della cartella attiva è " & ActiveWorkbook.Name
'Restituisce "prova1.xlsm"
MsgBox " Il nome di questa cartella di lavoro è " & ThisWorkbook.Name
'attiva "prova2.xlsm"
Workbooks("prova2.xlsm").Activate
'ritorna a "prova2.xlsm", mentre ThisWorkbook rimane su "prova1.xlsm"
MsgBox "Il nome della cartella attiva è " & ActiveWorkbook.Name
'ritorna a "prova1.xlsm"
MsgBox " Il nome di questa cartella di lavoro è " & ThisWorkbook.Name
'attiva "prova1.xlsm"
Workbooks("prova1.xlsm").Activate
'ritorna a "prova1.xlsm"
MsgBox "Active Workbook's name is " & ActiveWorkbook.Name
MsgBox "Il nome di questa cartella di lavoro è " & ThisWorkbook.Name
End Sub
```

Il Metodo Workbooks.Open

Si utilizza il metodo *Workbooks.Open* per aprire una cartella di lavoro con questa sintassi: *WorkbooksObject.Open(FileName, UpdateLinks, ReadOnly, Format, Password, WriteResPassword, IgnoreReadOnlyRecommended, Origin, Delimiter, Editable, Notify, Converter, AddToMru, Local, CorruptLoad)*
L'argomento *FileName* è necessario mentre tutti gli altri argomenti sono facoltativi, vediamo solo alcuni di loro. L'argomento FileName è un valore di tipo String che specifica il nome del file compresa l'estensione e il suo percorso, mentre l'argomento *UpdateLinks* se impostato non aggiornerà link o riferimenti esterni quando la cartella di lavoro è aperta, mentre specificando il valore 3 aggiornerà i link o riferimenti esterni.

L'argomento *ReadOnly* impostato a True apre la cartella di lavoro in modalità di sola lettura e l'argomento *password* è un valore stringa che specifica la password necessaria per aprire una cartella di lavoro protetta da password, omettendo tale argomento verrà richiesto all'utente una password.

L'argomento *WriteResPassword* è un valore stringa che specifica la password necessaria per aprire una cartella di lavoro in scrittura, (es. apertura di questo file senza la password renderà sola lettura), e omettendo tale argomento verrà richiesto all'utente una password. Si noti che quando una cartella di lavoro è aperta a livello di programmazione, le macro sono abilitate di default.

Metodo Workbooks.Add

Si utilizza il metodo Workbooks.Add per creare una nuova cartella di lavoro, che diventa anche la cartella di lavoro usando questa sintassi:*WorkbooksObject.Add (modello)*. Usando questo metodo VBA restituisce un oggetto cartella di lavoro. E 'opzionale specificare il *modello*, in quanto questo argomento è un valore stringa che specifica il nome (e il percorso) di un file di Excel esistente, e in questo caso il file specificato funge da modello per la nuova cartella di lavoro che viene creato, il che significa che la nuova cartella avrà lo stesso contenuto, formattazione, macro e la personalizzazione del file esistente che è specificato. È inoltre possibile definire una costante

- ***xlWBATTemplate Enumeration***: Per questo argomento, e in questo caso la cartella di lavoro appena creata conterrà un singolo foglio del tipo che è specificato
- ***xlWBATChart***: Creerà un foglio grafico
- ***xlWBATExcel4MacroSheet***: Crea una versione di Excel 4 con macro
- ***xlWBATExcel4IntlMacroSheet***: Crea una versione di Excel 4 con foglio macro internazionale
- ***xlWBATWorksheet***: Crea un foglio di lavoro.

Tralasciando l'argomento modello si creerà una nuova cartella di lavoro di Excel di default con tre fogli bianchi, in cui il numero predefinito dei fogli può essere modificato/impostato utilizzando la proprietà *Application.SheetsInNewWorkbook*

Il nome predefinito di una nuova cartella di lavoro quando viene creata utilizzando il metodo Add, e l'argomento modello viene omesso, è denominato*CartelN*, dove la prima cartella di lavoro sarà Cartel1, seguita da Cartel2, e così via, quando l'argomento modello è la costante *xlWBATWorksheet*, le cartelle di lavoro sono denominate SheetN e la prima cartella di lavoro creata sarà Foglio1, seguita da Foglio2, e così via

Esempio: Utilizzare il metodo Add per creare una nuova cartella di lavoro

```
Sub WorkbooksAdd()
Dim i As Integer, n As Integer
i = InputBox("Inserire il numero di cartelle da creare")
For n = 1 To i
Workbooks.Add
Next n
End Sub
```

Il Metodo Close

Si utilizza il metodo Close dell'oggetto Workbooks per chiudere tutte le cartelle di lavoro aperte con questa sintassi: *WorkbooksObject.Close* è inoltre possibile impostare la proprietà *DisplayAlerts* su False per non visualizzare alcuna richiesta o avviso alla chiusura di una cartella. Per chiudere tutte le cartelle di lavoro aperte, utilizzare la riga di codice *Workbooks.Close*, si può utilizzare il metodo Close dell'oggetto Workbook per chiudere una singola cartella di lavoro con questa sintassi: *WorkbookObject.Close (SaveChanges, filename, RouteWorkbook)*.

Tutti gli argomenti sono opzionali da specificare e tralasciando l'argomento SaveChanges verrà richiesto all'utente se salvare le modifiche, nel caso in cui siano state fatte, alla cartella di lavoro dopo l'ultimo salvataggio. Impostando l'argomento SaveChanges a True si salveranno tutte le modifiche apportate alla cartella di lavoro, e se impostato su False, la cartella di lavoro si chiude senza salvare le modifiche apportate e in entrambe le impostazioni la cartella di lavoro si chiude senza visualizzare alcuna richiesta per salvare le modifiche.

L'argomento *filename* dovrebbe essere utilizzato per specificare un nome di file per chiudere una cartella di lavoro che ancora non ha un nome di file associato, cioè una nuova cartella di lavoro, altrimenti con l'omissione di questo argomento si richiederà all'utente di specificare il nome del file prima di chiuderlo.
L'argomento nome del file può essere utilizzato come opzione se si desidera salvare una cartella di lavoro esistente, cioè che ha già un nome associato, con un nuovo nome di file.
L'argomento *RouteWorkbook* impostato a True consente di instradare o inviare al destinatario, se è presente una lista di distribuzione. Esempi di utilizzo di questo metodo:

Per chiudere la cartella di lavoro attiva dopo aver salvato le modifiche

```
ActiveWorkbook.Close SaveChanges: = True
'Per chiudere la cartella di lavoro "prova.xlsx", dopo aver salvato le modifiche
Workbooks ("prova.xlsx"). Close SaveChanges: = True
```

Esempio: Vari metodi di apertura e chiusura della cartella di lavoro con salvataggio

```
Sub WorkbookAdd()
Application.DisplayAlerts = False
ChDir ThisWorkbook.Path 'spostarsi dalla directory corrente a ThisWorkbook
Workbooks.Add "prova1.xlsx"
MsgBox " Il nome della Nuova cartella di lavoro e il numero di fogli è: " & ActiveWorkbook.Name & "; "
& ActiveWorkbook.Sheets.count
'salva come file xls di Excel 97, formato 2003
ActiveWorkbook.SaveAs fileName:=ActiveWorkbook.Name, FileFormat:=xlExcel8
ActiveWorkbook.ActiveSheet.Range("A1") = "xlsFile" 'Inserire in A1 del foglio attivo "xlsFile
ActiveWorkbook.Close SaveChanges:=True 'chiude la cartella dopo aver salvato le modifiche
Workbooks.Add 'Creare una nuova cartella di lavoro con tre fogli di lavoro
MsgBox " Il nome della Nuova cartella di lavoro e il numero di fogli è: " & ActiveWorkbook.Name & "; "
& ActiveWorkbook.Sheets.count
'salvare con estensione xlsx (2007)
ActiveWorkbook.SaveAs fileName:=ActiveWorkbook.Name, FileFormat:=xlOpenXMLWorkbook
ActiveWorkbook.ActiveSheet.Range("A1") = "xlsxFile" 'Inserire nella cella A1 del foglio "xlsxFile"
ActiveWorkbook.Close SaveChanges:=True 'chiude la cartella di lavoro dopo aver salvato le modifiche
Application.SheetsInNewWorkbook = 5 'Modificare il numero predefinito di fogli di lavoro a 5:
Workbooks.Add
MsgBox "Il nome della Nuova cartella di lavoro e il numero di fogli è: " & ActiveWorkbook.Name & "; "
& ActiveWorkbook.Sheets.count
ActiveWorkbook.SaveAs fileName:="NewWorkbookSaved.xlsm",  'salvare la nuova cartella come xlsm,
FileFormat:=xlOpenXMLWorkbookMacroEnabled
ActiveWorkbook.ActiveSheet.Range("A1") = "xlsmFile" 'Inserire " xlsmFile " in A1 del foglio attivo
ActiveWorkbook.Close SaveChanges:=True 'chiude la cartella di lavoro dopo aver salvato le modifiche
Workbooks.Add (xlWBATWorksheet) 'Creare una nuova cartella di lavoro con un foglio di lavoro
MsgBox " Il nome della Nuova cartella di lavoro e il numero di fogli è: " & ActiveWorkbook.Name & "; "
& ActiveWorkbook.Sheets.count
ActiveWorkbook.SaveAs fileName:=ActiveWorkbook.Name 'salva la cartella, omettendo il formato file
ActiveWorkbook.ActiveSheet.Range("A1") = "DefaultFileFormat" 'Inserisce "DefaultFileFormat" in A1
ActiveWorkbook.Close SaveChanges:=True 'chiude la cartella di lavoro dopo aver salvato le modifiche
Application.DisplayAlerts = True
End Sub
```

Il Metodo Workbook.SaveCopyAs

Si utilizza il metodo Workbook.SaveCopyAs per salvare una copia della cartella di lavoro e questo metodo non modifica la cartella di lavoro aperta con questa sintassi: *WorkbookObject.SaveCopyAs (filename)*.
L'argomento *filename* è facoltativo e viene utilizzato per specificare il nome del file con cui viene memorizzata la copia della cartella di lavoro. Si noti che il metodo Workbook.SaveCopyAs permette di salvare con lo stesso formato di file, cioè la cartella di lavoro e la sua copia devono avere lo stesso formato di file, altrimenti durante l'apertura della copia salvata otterrete un messaggio - "Il file che si sta cercando di aprire è in un formato diverso da quello specificato dall'estensione del file ... " ed è possibile che il file non possa venire aperto.

Esempio: Utilizzare il metodo Workbook.SaveCopyAs per salvare una copia della cartella di lavoro attiva

```
Sub WorkbookSaveCopyAs1()
Dim LastRow As Long
'Determinare l'ultima riga scritta nella colonna "A"
LastRow = ActiveWorkbook.ActiveSheet.Range("A" & Rows.count).End(xlUp).Row
'Salvare una copia della cartella di lavoro se i dati in una colonna del foglio attivo è
maggiore di un numero specifico di righe
If LastRow >= 100 Then
'Salvare una copia della cartella di lavoro attiva specificando un nome di file
ActiveWorkbook.SaveCopyAs "C:\Document\Test\Copia_1.xlsm"
End If
'La cartella di lavoro corrente rimane la cartella attiva, la copia è stata salvata e chiusa
MsgBox ActiveWorkbook.Name
End Sub
```

Esempio: Utilizzare il metodo Workbook.SaveCopyAs per salvare una copia di ThisWorkbook con un nome univoco ogni volta.

```
Sub WorkbookSaveCopyAs2()
Dim fname As String, extn As String, MyStr As String
Dim i As Integer, lastDot As Integer
'cambiare la directory corrente a ThisWorkbook
ChDir ThisWorkbook.Path
'Trovare la posizione del punto per distinguere l'estensione del file
For i = 1 To Len(ThisWorkbook.Name)
If Mid(ThisWorkbook.Name, i, 1) = "." Then
lastDot = i
End If
Next i
'Estensione del file estratta e dot prima estensione
extn = Right(ThisWorkbook.Name, Len(ThisWorkbook.Name) - lastDot + 1)
'nome della cartella estratto escluso l'estensione dal punto
MyStr = Left(ThisWorkbook.Name, lastDot - 1)
'specificare il nome per la copia. Parte del nome del file renderà il nome univoco
fname = MyStr & "_" & Format(Now(), "yyyy-mm-dd hh-mm-ss AMPM") & extn
'salvare una copia di ThisWorkbook specificando un nome di file
ThisWorkbook.SaveCopyAs fname
'la cartella di lavoro corrente rimane la cartella attiva, la copia salvata e chiusa
MsgBox ActiveWorkbook.Name
End Sub
```

Esempio: Utilizzare il metodo Workbook.SaveCopyAs per salvare una copia della cartella di lavoro specificata.

```
Sub WorkbookSaveCopyAs3()
'Aprire una cartella di lavoro specificata nella directory corrente
Workbooks.Open "C:\Documenti\Test\prova_3.xlsx"
'la cartella di lavoro aperta diventa la cartella attiva di lavoro
MsgBox ActiveWorkbook.Name
'salvare una copia della cartella di lavoro aperta, specificando un nome di file
ActiveWorkbook.SaveCopyAs "C:\Documenti\Test\Copia_di_prova3.xlsx"
'la cartella di lavoro aperta rimane la cartella di lavoro attiva, la copia rimane chiusa
MsgBox ActiveWorkbook.Name
'chiudere la cartella di lavoro aperta - notare che né la cartella di lavoro aperta o la
copia salvata saranno aperti dopo questo comando
ActiveWorkbook.Close
End Sub
```

Il metodo Workbook.Save

Si utilizza il metodo Workbook.Save per salvare una cartella di lavoro specificata con questa sintassi: *WorkbookObject.Save*. Esempio: Salvare la cartella di lavoro denominato "prova_1.xlsm":

```
Workbooks ("prova_1.xlsm"). Save
```

Si utilizza l'argomento *SaveChanges* del metodo *Close* dell'oggetto Workbook, per salvare la cartella di lavoro prima della chiusura. Esempio per chiudere la cartella di lavoro dopo averla salvata:

```
Workbooks(Workbooks.Count).Close SaveChanges:=True
```

Si utilizzare il metodo Workbook.SaveAs per salvare le modifiche della cartella di lavoro in un file separato con questa sintassi: *WorkbookObject.SaveAs (FileName, FileFormat, password, WriteResPassword, ReadOnlyRecommended, CreateBackup, AccessMode, ConflictResolution, AddToMru, TextCodepage, TextVisualLayout , locale)*. Tutti gli argomenti sono opzionali, vediamo solo alcuni di loro.
L'argomento *FileName* è un valore di tipo String che specifica il nome del file compresa l'estensione e il suo percorso, omettendo il percorso il file verrà salvato nella directory corrente, mentre *FileFormat* specifica il formato del file da salvare, il formato predefinito è la versione corrente di Excel, mentre per un file esistente l'impostazione predefinita è il formato del file che era specificato. Si ricorda che in Excel 2007-2010 durante l'utilizzo SaveAs, è necessario specificare il parametro FileFormat per salvare un nome di file con estensione Xlsm se la cartella di lavoro in fase di salvataggio non è un file Xlsm, vale a dire:

```
FileFormat: = xlOpenXMLWorkbookMacroEnabled
```

È possibile specificare una password case-sensitive fino a 15 caratteri per l'apertura del file, utilizzando l'argomento *password* specificando una password per aprire un file, se non si inserisce la password si aprirà in sola lettura utilizzando l'argomento *WriteResPassword* per specificare la password.

Esempio: Salvare una cartella di lavoro esistente con un nuovo nome e formato

```
Sub workbookSaveAs1()
'salvare la cartella di lavoro con indice 2 e con un nuovo nome e formato del file.
'ricordate che tutti i file dovrebbero essere aperti in una singola istanza di Excel.
'Ricordate che in Excel 2007-2010 durante l'utilizzo di SaveAs, è necessario specificare il
'parametro FileFormat per salvare un nome di file con estensione xlsm se la cartella di
'lavoro non è un file xlsm
Workbooks(2).SaveAs "Workbook_cambiato.xlsm"
End Sub
```

Esempio: Salvataggio di una nuova cartella di lavoro:

```
Sub workbookSaveAs2()
'aggiungere una nuova cartella di lavoro
Workbooks.Add
'salvare la nuova cartella di lavoro, che diventa la cartella di lavoro attiva, protetta da
password
ActiveWorkbook.SaveAs fileName:="Nuova_cart.xlsx", Password:="abc123"
'salvare la nuova cartella di lavoro come un file xlsm con password di protezione
'ActiveWorkbook.SaveAs fileName:="Nuova_cart.xlsm",
'FileFormat:=xlOpenXMLWorkbookMacroEnabled, Password:="abc123"
'salvare la nuova cartella di lavoro come file xlsm con password di protezione, ma può
'essere aperta in sola lettura senza fornire la password
ActiveWorkbook.SaveAs fileName:="Nuova_cart.xlsm",
FileFormat:=xlOpenXMLWorkbookMacroEnabled, WriteResPassword:="abc123"
End Sub
```

Si utilizza la proprietà Workbook.Saved per determinare se sono state apportate modifiche alla cartella di lavoro dopo l'ultimo salvataggio con questa sintassi: *WorkbookObject .Saved* che restituisce un valore booleano, dove True indica che la cartella di lavoro non è stata modificata dopo l'ultimo salvataggio. L'impostazione di questa proprietà su True prima di chiudere una cartella di lavoro in cui sono state apportate modifiche non chiederà di salvare la cartella di lavoro e non verranno salvate le modifiche.

Esempio: Impostare la proprietà Workbook.Saved a True per chiudere la cartella di lavoro con nessuna richiesta di salvare le modifiche:

```
Sub workbookSaved()
'Aprire una cartella di lavoro
Workbooks.Open "C:\Documents\prova5.xlsx"
'Workbook.Saved restituirà true a indicare che non sono state apportate modifiche alla
cartella di lavoro dopo che è stata salvata l'ultima volta
MsgBox Workbooks("prova5").Saved
'modifichiamo la cartella di lavoro
Workbooks("prova5.xlsx").ActiveSheet.Range("A1") = "Ciao Mondo!"
'Workbook.Saved restituirà False per indicare che le modifiche sono state apportate alla
'cartella di lavoro dopo che è stata salvata l'ultima volta
MsgBox Workbooks("prova5.xlsx").Saved
'impostare la proprietà Workbook.Saved su true
Workbooks("prova5.xlsx").Saved = True
'chiudere la cartella di lavoro con nessuna richiesta di salvare le modifiche e questo NON
salverà le eventuali modifiche. Se la proprietà Saved non è stata impostata su True si
otterrà un prompt per salvare le modifiche
Workbooks("prova5.xlsx").Close
End Sub
```

Metodo Workbook.Activate

Si utilizza il metodo Workbook.Activate per attivare una cartella di lavoro e se la cartella di lavoro dispone di più finestre, il metodo attiva la prima finestra. Sintassi: *WorkbookObject.Activate* . Per attivare la cartella di lavoro denominato "prova5.xlsx", utilizzare il codice

```
Workbooks ("prova5.xlsx"). Activate
```

Metodo Workbook.PrintPreview

Si utilizza il metodo Workbook.PrintPreview per visualizzare un'anteprima di come verrà stampata la cartella con questa sintassi:*WorkbookObject.PrintPreview (EnableChanges)*. E 'facoltativo specificare l'argomento EnableChanges, che accetta un valore booleano (il valore predefinito è True), per consentire o non consentire all'utente di modificare le opzioni di impostazione della pagina (es. orientamento della pagina, il ridimensionamento, i margini, ecc) disponibili in anteprima di stampa.

Esempio: Utilizzo di PrintPreview

```
Sub PrintPreview()
'Anteprima di stampa del foglio attivo della cartella di lavoro, impedendo all'utente di
'cambiare le impostazione della pagina di anteprima di stampa.
Workbooks("prova5.xlsx").PrintPreview EnableChanges:=False
'anteprima di stampa di "Foglio3" di "prova5.xlsx, permettendo all'utente di cambiare le
'impostazioni di pagina disponibili in anteprima di stampa.
Workbooks("prova5.xlsx").Worksheets("Foglio3").PrintPreview EnableChanges:=True
End Sub
```

Metodo Workbook.SendMail

Molte persone utilizzano Outlook come client per la posta elettronica, è possibile automatizzare Outlook, consentendo maggiori funzionalità di inviare e-mail, lavorando con gli oggetti di Outlook utilizzando VBA in Excel. L'automazione è un processo mediante il quale una applicazione comunica o controlla con un'altra applicazione e un'opzione per inviare e-mail da Excel è quella di utilizzare il metodo Workbook.SendMail. Con il metodo Workbook.SendMail , si utilizza il sistema di posta installato per inviare una cartella di lavoro. Sintassi: *WorkbookObject.SendMail(Recipients, Subject, ReturnReceipt))*.

L'argomento *Recipients* è necessario per specificare un singolo destinatario o più destinatari come testo o un array di stringhe di testo, rispettivamente e l'argomento *Subject* è facoltativo e viene utilizzato per specificare l'oggetto della mail e se si omette questo argomento di default verrà usato il nome della cartella di lavoro come oggetto. L'argomento *ReturnReceipt* è facoltativo, se viene usato si specifica il suo valore, che può essere True o False e indica la richiesta di richiedere una ricevuta di ritorno, il valore predefinito è False. Per inserire il nome del destinatario come testo:

```
ActiveWorkbook.SendMail Recipients:="nome_destinatario"
```

Esempio: Inviare una cartella di lavoro, specificando il nome del destinatario o indirizzo email

```
Sub inviaM_1()
Workbooks.Open ("C:\Documents\Test1\prova1.xlsx")
'specificare il nome del destinatario come testo
ActiveWorkbook.SendMail Recipients:="Gino Primo", Subject:="Ciao",
ReturnReceipt:=True
'specifica indirizzo e-mail di destinazione
ActiveWorkbook.SendMail Recipients:="info@gino.com", Subject:="Ciao",
ReturnReceipt:=True
End Sub
```

Esempio: Inviare cartella di lavoro, specificando più destinatari.

```
Sub inviaM_2()
'Specificare più destinatari
ThisWorkbook.SendMail Array("Gino Primo", "Beppe Secondo")
'specificare più indirizzi e-mail come destinazione
ThisWorkbook.SendMail Array(" info@gino.com", " info@beppe.com")
End Sub
```

Esempio: Inviare un foglio Excel con il metodo Sendmail.

```
Sub inviaM_3()
Dim name1 As String, name2 As String
Dim wb As Workbook
'cambiare la directory corrente a ThisWorkbook, utilizzando ChDir
ChDir ThisWorkbook.Path
'crea una nuova cartella di lavoro con un unico foglio da copiare da ThisWorkbook
ThisWorkbook.Worksheets("Foglio1").Copy
'impostare la variabile wb per la nuova cartella di lavoro, che diventa la cartella attiva
Set wb = ActiveWorkbook
'rinominare il foglio nella nuova cartella di lavoro
wb.Sheets("Foglio1").Name = "Nuovo Foglio"
'salvare la nuova cartella di lavoro con un nome di file, nella cartella predefinita
wb.SaveAs fileName:="NuovoF.xlsx"
'assegnare delle variabili per I destinatari
name1 = "Gino Primo"
name2 = "Beppe Secondo"
'Inserire i nomi dei destinatari in una matrice
wb.SendMail Array(name1, name2),
Subject:=ThisWorkbook.Worksheets("Foglio1").Range("A1"), ReturnReceipt:=False
'chiudere la nuova cartella di lavoro
wb.Close
End Sub
```

Esempio: Inviare la cartella di lavoro, specificando il destinatario in una cella del foglio di lavoro, con l'indicazione della data in oggetto.

```
Sub inviaM_4()
Dim strRec As String
'La cella A14 contiene: info@gino.com
strRec = ThisWorkbook.Sheets("Foglio1").Range("A14").Value
'l'oggetto apparirà come "Si prega di controllare 10/08/2014 10:43:06 "
ActiveWorkbook.SendMail Recipients:=strRec, Subject:="Si prega di controllare " &
Format(Now, "dd/mm/yyyy hh:mm:ss AMPM")
End Sub
```

Esempio: Inviare la cartella di lavoro, specificando più destinatari da intervallo di prospetto.

```
Sub inviaM_5()
Dim MyArr As Variant
'la cella A14 contiene info@gino.com, e la cella A15 contiene info@beppe.com
ActiveWorkbook.SendMail
Recipients:=ThisWorkbook.Sheets("Foglio1").Range("A14:A15").Value
MyArr = ThisWorkbook.Sheets("Foglio1").Range("A14:A15")
ActiveWorkbook.SendMail Recipients:=MyArr
'La cella A10 contiene gino Primo la cella A11 contiene beppe secondo, in A14 è stato
inserito info@gino.com, e in A15 info@beppe.com
'Gli intervalli sono stati nominati "nomi_1"=Range("A10:A11"),
"email_1"=Range("A14:A15")
MyArr = ThisWorkbook.Sheets("Sheet1").Range("nomi_1")
MyArr = ThisWorkbook.Sheets("Sheet1").Range("email_1")
ActiveWorkbook.SendMail Recipients:=MyArr
End Sub
```

Proprietà Workbook.ActiveSheet

Questa proprietà restituisce la scheda attualmente attiva in una cartella di lavoro e si usa con questa sintassi: *WorkbookObject.ActiveSheet*. Se appaiono più finestre per una cartella di lavoro, ActiveSheet potrebbe essere diversa per ogni finestra, se non c'è un foglio attivo, questa proprietà restituisce Nothing. Utilizzare il metodo Activate dell'oggetto foglio di lavoro per attivarne una, cioè attivare un foglio. Per esempio

```
ActiveWorkbook.Sheets ("Foglio3"). Activate
```

Attiverà il foglio denominato "Foglio3" nella cartella di lavoro attiva, il cui nome verrà visualizzato da

```
MsgBox ActiveWorkbook.ActiveSheet.Name
```

Proprietà Workbook.Name

Questa proprietà restituisce il nome di una cartella di lavoro (valore stringa).
Sintassi: *WorkbookObject.Name* ed è di sola lettura e non è possibile modificare il nome della cartella di lavoro. È tuttavia possibile utilizzare il metodo Workbook.SaveAs per salvare una cartella di lavoro esistente (in un nuovo file) con un nuovo nome o salvare una nuova cartella di lavoro con nome.

```
Sub WorkbookName ()
'restituire i nomi di tutte le cartelle di lavoro aperte
 Dim i As Integer, conta As Integer
'restituisce il numero delle cartelle di lavoro
count = Workbooks.Count
'Restituisce i nomi di tutte le cartelle
For i = 1 To count
MsgBox Workbooks(i).Name
Next i
End Sub
```

Proprietà Workbook.ActiveChart

Questa proprietà restituisce il grafico attualmente attivo, che può essere un foglio grafico o un grafico incorporato. Sintassi: *WorkbookObject.ActiveChart*, se non c'è un grafico attivo, questa proprietà restituisce Nothing. Si utilizza il metodo Activate della 'oggetto Chart' o 'oggetto ChartObject' per attivare rispettivamente un foglio grafico o un grafico incorporato, tenendo presente che un foglio grafico è attivo se selezionato dall'utente o attivato utilizzando il metodo Activate. Vedi esempio sotto riportato

Esempio: Illustrare proprietà ActiveChart e metodo Activate.

```
Sub attiva_graf()
ActiveWorkbook.Sheets("Chart1").activate 'Attivare il foglio grafico denominato "Chart1"
'In alternativa, per attivare un foglio grafico: 'ActiveWorkbook.Charts (1). activate
'attivare il grafico incorporato denominato "Grafico 13" in "Foglio2", utilizzando il metodo Activate
ActiveWorkbook.Sheets("Foglio2").ChartObjects("Chart 13").activate
'In alternativa, per attivare il grafico 1 nel "Foglio2" ActiveWorkbook.Sheets ("Foglio2")
ChartObjects (1).Activate, restituisce il grafico attivo - "Foglio2 Chart 13"
MsgBox ActiveWorkbook.ActiveChart.Name
'Aggiungere un titolo al grafico attivo (grafico incorporato denominato "Chart 13" in "Foglio2"):
With ActiveWorkbook.ActiveChart
.HasTitle = True
.ChartTitle.Text = "Ems_Grafico1"
End With
MsgBox ActiveWorkbook.ActiveChart.ChartTitle.Text 'Restituisce il titolo del grafico attivo
End Sub
```

Proprietà Workbook.FileFormat

Questa proprietà restituisce il formato del file della cartella di lavoro. Sintassi: *WorkbookObject.FileFormat* ed è di sola lettura, inoltre è possibile specificare o impostare il formato di file durante il salvataggio di una cartella di lavoro esistente o nuova utilizzando il metodo Workbook.SaveAs (discusso sopra).

Proprietà Workbook.Path

Si utilizza questa proprietà per restituire il percorso (valore stringa) completa della cartella di lavoro Sintassi: *WorkbookObject.Path*.

Proprietà Workbook.Password

Si utilizza questa proprietà per impostare o restituire una password per l'apertura di una cartella di lavoro e si tratta di una proprietà di lettura/scrittura con la seguente sintassi: *WorkbookObject.Password*. Vedere l'esempio sottostante che illustra come impostare una password per aprire una cartella di lavoro e modificare e cancellare una password esistente.

Esempio: Imposta una password per la cartella di lavoro da aprire, cambiare password, eliminare password.

```
Sub pass_cart()
Application.DisplayAlerts = False
Dim fpath As String, fname As String, Pswd As String, newPswd As String
'specificare il nome completo (cioè il percorso e nome) del file da aprire
fpath = "C:\Documents\Test1"
fname = "prova1.xlsx"
'specificare la password per aprire il file
Pswd = "123"
newPswd = "abc"

'aprire la cartella di lavoro senza password
Workbooks.Open fileName:=(fpath & "\" & fname)
'imposta password
ActiveWorkbook.Password = Pswd
'chiudi la cartella salvando le modifiche
ActiveWorkbook.Close SaveChanges:=True

'apri la cartella di lavoro con una password e aggiorna link o riferimenti esterni
Workbooks.Open fileName:=(fpath & "\" & fname), UpdateLinks:=3, Password:=Pswd
'il file verrà salvato senza visualizzare alcuna richiesta
ActiveWorkbook.SaveAs fileName:=(fpath & "\" & fname), Password:=newPswd
'chiudi la cartella salvando le modifiche
ActiveWorkbook.Close SaveChanges:=True

'apri la cartella di lavoro con una password e aggiorna link o riferimenti esterni
Workbooks.Open fileName:=(fpath & "\" & fname), UpdateLinks:=3,
Password:=newPswd
'il file verrà salvato senza visualizzare alcuna richiesta
ActiveWorkbook.SaveAs fileName:=(fpath & "\" & fname), Password:=""
'chiudi la cartella salvando le modifiche
ActiveWorkbook.Close SaveChanges:=True
Application.DisplayAlerts = True
End Sub
```

Metodi e Proprietà per gestire le righe del foglio di lavoro

La Proprietà Range.End

In VBA sarà spesso necessario fare riferimento a una cella alla fine di un blocco, ad esempio, per determinare l'ultima riga utilizzata in un intervallo. La struttura finale è utilizzata con riferimento a un oggetto *Range* e restituisce la cella che si trova alla fine della regione in cui il range di riferimento è contenuto in una determinata direzione, ed è simile a premere CTRL + freccia SU', CTRL + freccia GIÙ, CTRL + freccia Sinistra o CTRL + freccia Destra. La sintassi è la seguente:

RangeObject. End (Direction).

È necessario specificare l'argomento **Direction**, che indica la direzione di movimento, per esempio *.End (xlDown)* indica lo spostamento verso il basso, mentre *.End (xlToRight)* indica lo spostamento verso destra.

Utilizzare End (xlUp) per determinare l'ultima riga con i dati in una colonna

End (xlUp) è uno dei metodi più comunemente utilizzati per determinare l'ultima riga utilizzata, contenente dei dati.Rows.Count restituisce l'ultima riga del foglio di lavoro, se consideriamo che Excel 2007 dispone di 1.048.576 righe, l'istruzione *.Cells (Rows.Count, "B")* restituisce la cella B1048576, vale a dire l'ultima cella della colonna B, e il codice parte da questa cella e scorre tutta la colonna verso l'alto fino a trovare una cella che contiene dei dati

```
Sub ultima_1()
Dim ultimaR As Long
ultimaR = ActiveSheet.Cells(Rows.Count, "B").End(xlUp).Row
MsgBox ultimaR
End Sub

Sub ultima_2()
Dim ultimaR As Long
ultimaR = ActiveSheet.Range("B" & Rows.Count).End(xlUp).Row
MsgBox ultimaR
End Sub
```

Utilizzare End (xlToLeft) per determinare l'ultima colonna con i dati

Restituisce il numero dell'ultima colonna con i dati in una riga specificata, nel caso di una riga vuota restituirà il valore 1. Non vengono considerate le celle formattate, ma senza dati, mentre si considerano costanti e formule. Se l'ultima colonna con i dati è nascosta, questa colonna viene ignorata

```
Sub ultima_3()
Dim ultimaC As Integer
ultimaC = ActiveSheet.Cells(2, Columns.Count).End(xlToLeft).Column
MsgBox ultimaC
End Sub
```

La proprietà UsedRange per trovare l'ultima riga

Per restituire l'intervallo utilizzato in un foglio di lavoro, si utilizza la proprietà *Worksheet.UsedRange* che presenta la seguente sintassi: *WorksheetObject.UsedRange*

E include anche le celle formattate con dati o celle con dati il cui contenuto è stato eliminato, e in questo caso potrebbe includere apparentemente celle vuote visibili. Ad esempio, se si applica il formato data a una cella, in questo caso, cancellare il contenuto e la formattazione potrebbe non essere sufficiente per re-impostare la riga o cella e in questo caso si dovrà eliminare la riga.

```
Sub ultima_4()
Dim ultimaR As Long
ultimaR = ActiveSheet.UsedRange.Row - 1 + ActiveSheet.UsedRange.Rows.Count
MsgBox ultimaR
End Sub

Sub ultima_5()
Dim ultimaR As Long
ultimaR = ActiveSheet.UsedRange.Rows(ActiveSheet.UsedRange.Rows.Count).Row
MsgBox ultimaR
End Sub
```

La Proprietà UsedRange per trovare l'ultima colonna

```
Sub ultima_6()
Dim ultimaC As Integer
ultimaC = ActiveSheet.UsedRange.Column - 1 + ActiveSheet.UsedRange.Columns.Count
MsgBox ultimaC
End Sub

Sub ultima_7()
Dim ultimaC As Integer
ultimaC =
ActiveSheet.UsedRange.Columns(ActiveSheet.UsedRange.Columns.Count).Column
MsgBox ultimaC
End Sub
```

La Proprietà UsedRange per contare il numero di righe utilizzate

```
Sub prova_RU()
Dim rigaU As Long
rigaU = ActiveSheet.UsedRange.Rows.Count
MsgBox rigaU
End Sub
```

La Proprietà UsedRange per contare il numero di colonne utilizzate

```
Sub prova_CU()
Dim colonnaU As Integer
colonnaU = ActiveSheet.UsedRange.Columns.Count
MsgBox colonnaU
End Sub
```

La proprietà UsedRange per trovare la prima riga utilizzata

```
Sub prima_RU1()
Dim primaR As Long
primaR = ActiveSheet.UsedRange.Cells(1).Row
MsgBox primaR
End Sub

Sub prima_RU2()
Dim primaR As Long
primaR = ActiveSheet.UsedRange.Row
MsgBox primaR
End Sub
```

La Proprietà UsedRange per trovare la prima colonna utilizzata

```
Sub prima_CU1()
Dim primaC As Integer
primaC = ActiveSheet.UsedRange.Cells(1).Column
MsgBox primaC
End Sub

Sub prima_CU2()
Dim primaC As Integer
primaC = ActiveSheet.UsedRange.Column
MsgBox primaC
End Sub
```

La Proprietà Row e Column

Per restituire il numero della prima riga in un intervallo, si utilizza la proprietà *Range.Row* e se l'intervallo specificato contiene più aree, questa proprietà restituirà il numero della prima riga della prima area. La sintassi utilizzata è la seguente: *RangeObject.Row*
Mentre invece per restituire il numero della prima colonna in un intervallo, si utilizza la proprietà *Range.Column* e se l'intervallo specificato contiene più aree, questa proprietà restituirà il numero della prima colonna nella prima area. La sintassi è: *RangeObject.Column*

Esempi della proprietà Row
Applicare il colore giallo a tutte le righe dell'intervallo B2:D4
Worksheets ("Foglio1"). Range ("B2: D4"). Rows.Interior.Color = vbYellow

Applicare il colore verde alla prima riga del range B2:D2
Worksheets ("Foglio1"). Range ("B2: D4"). Row (1) Interior.Color = vbGreen

Se l'oggetto specificato contiene più zone, le righe della prima area verranno restituite solo da questa proprietà.
Prendiamo l'esempio di 2 aree nell'intervallo specificato, la prima area sarà "B2: D4" e la seconda sarà "F3: G6", il seguente codice applica il colore rosso alle celle alla prima riga della prima area (B2:D4)

Worksheets ("Foglio1") .Range ("B2: D4, F3: G6"). Row (1) Interior.Color = vbRed

Esempi della proprietà Columns
Applicare il colore giallo a tutte le colonne dell'intervallo specificato, cioè B2:D4
Worksheets ("Foglio1"). Range ("B2: D4"). Columns.Interior.Color = vbYellow

Applicare il colore verde alla prima colonna del range B2:B4
Worksheets ("Foglio1"). Range ("B2: D4"). Columns (1) Interior.Color = vbGreen

Se l'oggetto specificato contiene più zone, le colonne della prima area verranno restituite solo da questa proprietà. Per esempio se abbiamo 2 aree nell'intervallo specificato, e la prima area sarà "B2: D4" mentre la seconda area sarà "F3: G6", il seguente codice applica il colore rosso alle celle dalla prima colonna della prima area alle celle da B2 a B4

Worksheets ("Foglio1"). Range ("B2: D4, F3: G6"). Columns (1) Interior.Color = vbRed

Utilizzare la proprietà End (xlDown) per determinare l'ultima riga

```
Sub ultimaR1()
Dim ultimaR As Long
ultimaR = ActiveSheet.Range("D2").End(xlDown).Row
MsgBox ultimaR
End Sub
```

Utilizzare End (xlToRight) per determinare l'ultima colonna

```
Sub ultima8()
Dim ultimaC As Integer
ultimaC = ActiveSheet.Range("C4").End(xlToRight).Column
MsgBox ultimaC
End Sub
```

Esempi di utilizzo della proprietà Range.End: Selezione di una particolare riga o colonna come da Figura 1

	A	B	C	D	E	F	G	H	I	J
1										
2	55	66	12	245	Pietro		Mario	22		
3	45		14	15	11			Sara	245	12
4	35		5	5	5					
5	25		7	7						
6	15		6	8	Luca					
7	5		45	8	7		22	24	26	
8	-5		56	8	7					
9	-15		125	8	7			1		
10	-25		4	8	7					
11			66		7			2		
12			Elena		7			3		
13					7			4		
14				1	7			5	7	9
15				2	7					
16				3	7		33	6	Mario	Silvia
17			55	4	7					
18			66		7		34			
19				5	7		35			
20				6	6					

Fig. 1

```
Sub prova1()
Dim ws As Worksheet
Set ws = Worksheets("Foglio1")
ws.activate
 Range("C5").End(xlDown).Select 'seleziona la cella C12 (Elena)
'seleziona la cella C17 (55), la cella C12 è l'ultima cella di dati in un blocco
'in questo caso si seleziona la cella successiva con dati che è C17
Range("C12").End(xlDown).Select
Range("C17").End(xlDown).Select 'seleziona la cella C18 (66)
'seleziona la cella C17 (55), la cella C14 è una cella vuota
'e in questo caso seleziona la cella successiva con i dati
Range("C14").End(xlDown).Select
'seleziona l'ultima riga del foglio di lavoro se la colonna è vuota
'che è la cella F1048576 in quanto Excel 2007 ha 1.048.576 righe
Range("F1").End(xlDown).Select
Range("C7").End(xlToRight).Select 'seleziona la cella E7 (7)
Range("E7").End(xlToRight).Select 'seleziona la cella G7 (22)
'seleziona cella XFD7, che è l'ultima colonna della riga 7, in quanto
'la cella I7 è l'ultima cella con i dati in questa riga
Range("I7").End(xlToRight).Select
Range("I14").End(xlUp).Select 'seleziona la cella I7 (26)
Range("E18").End(xlUp).Select 'seleziona la cella E6 (Luca)
Range("C5", Range("C5").End(xlDown)).Select 'seleziona il range C5:C12
End Sub
```

Il metodo Find per determinare l'ultima riga

Restituisce l'ultima riga con i dati in un foglio di lavoro. In caso di un foglio di lavoro vuoto darà un errore di run-time. Per cercare un articolo specifico o un valore in un intervallo, si utilizza il metodo Find che restituisce il Range, vale a dire, la prima cella, dove si trova l'elemento o valore. Se non viene trovata alcuna corrispondenza, restituisce Nothing.

L'istruzione SearchDirection

È possibile specificare l'argomento *xlNext* per indicare di eseguire ricerche verso il basso (cioè al valore corrispondente successivo) o *xlPrevious* per ricerche verso l'alto o all'indietro (cioè al valore corrispondente precedente) nel campo di ricerca. Il valore predefinito è xlNext. Se si specifica *After: = Range ("A13")*, in cui il campo di ricerca è il Range ("A1: A20") e si imposta la direzione di ricerca in *SearchDirection: = xlNext*, allora la funzione di ricerca inizierà a cercare dalla cella A14 fino alla A20 per poi ricercare dal Range ("A1") fino al Range ("A13")

```
Sub prova2()
Dim ultimaC As Long, rng As Range
Set rng = ActiveSheet.Cells
ultimaC = rng.Find(What:="*", After:=rng.Cells(1), Lookat:=xlPart, LookIn:=xlFormulas,
SearchOrder:=xlByRows, SearchDirection:=xlPrevious, MatchCase:=False).Row
MsgBox ultimaC
End Sub
```

Il metodo Find per determinare l'ultima colonna

```
Sub prova3()
Dim ultimaC As Integer, rng As Range
Set rng = ActiveSheet.Cells

ultimaC = rng.Find(What:="*", After:=rng.Cells(1), Lookat:=xlPart, LookIn:=xlFormulas,
SearchOrder:=xlByColumns, SearchDirection:=xlPrevious, MatchCase:=False).Column

MsgBox ultimaC
End Sub
```

Il metodo SpecialCells per trovare l'ultima riga

```
Sub prova4()
Dim ultimaR As Long
ultimaR = ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Row
MsgBox ultimaR
End Sub

Sub prova5()
Dim ultimaR As Long
ultimaR = ActiveSheet.Range("A1").SpecialCells(xlCellTypeLastCell).Row
MsgBox ultimaR
End Sub
```

Il Metodo Range.SpecialCells

Si utilizza il metodo Range.SpecialCells con la sintassi: RangeObject.SpecialCells (Type, Value), dove l'argomento *type* specifica il tipo di cella come costanti*XlCellType*, da restituire ed è obbligatorio specificare questo argomento, mentre invece l'argomento *Value* è facoltativo e specifica i valori come per le costanti *XlSpecialCellsValue*, nel caso di *xlCellTypeConstants* o *xlCellTypeFormulas* viene specificato nell'argomento Type. Non specificando l'argomento Value per impostazione predefinita vengono inclusi tutti i valori delle costanti o formule, nel caso di xlCellTypeConstants o xlCellTypeFormulas rispettivamente. Usando questo metodo viene restituito un oggetto Range, composto da celle corrispondenti agli argomenti type e value specificati

I vari tipi di Costanti XlCellType

- xlCellTypeAllFormatConditions: Si riferisce a tutte le celle con formattazione condizionale (valore -4172)
- xlCellTypeAllValidation: Fa riferimento alle celle che contengono una convalida (valore -4.174)
- xlCellTypeBlanks: Si riferisce a celle vuote (valore 4)
- xlCellTypeComments: Fa riferimento alle celle con commenti (valore di -4144)
- xlCellTypeConstants: Fa riferimento alle celle che contengono costanti (valore 2)
- xlCellTypeFormulas: Fa riferimento alle celle che contengono formule (valore -4.123)
- xlCellTypeLastCell: Si riferisce all'ultima cella nell'intervallo utilizzato (valore 11)
- xlCellTypeSameFormatConditions: Si riferisce a celle con lo stesso formato (valore -4173)
- xlCellTypeSameValidation: Si riferisce a celle con la stessa convalida (valore -4175)
- xlCellTypeVisible: Si riferisce a tutte le celle che sono visibili (valore 12)

Il Metodo SpecialCells per trovare l'ultima colonna

```
Sub prova6()
Dim ultimaC As Integer
ultimaC = ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Column
MsgBox ultimaC
End Sub

Sub prova7()
Dim ultimaC As Integer
ultimaC = ActiveSheet.Range("A1").SpecialCells(xlCellTypeLastCell).Column
MsgBox ultimaC
End Sub
```

Oggetto Application – Metodi e Proprietà

L'oggetto *Application* di Excel rappresenta l'applicazione Microsoft Excel, definita anche applicazione Host. Se l'applicazione Host è Microsoft Word, l'oggetto Application si riferisce e rappresenta l'applicazione Word. Excel presuppone che anche quando non è specificato, il qualificatore dell'oggetto Application non è necessario per essere utilizzato nel codice VBA, perché l'applicazione di default è Excel stesso, a meno che non si vuole fare riferimento ad altre applicazioni esterne (come Microsoft Word o Access) nel codice o se si vuole fare riferimento a Excel da un'altra un'applicazione come Microsoft Word.

Tutte le applicazioni Microsoft Office, che utilizzano VBA, hanno un proprio modello di oggetti e durante la scrittura di codice VBA in Microsoft Excel, si prevede di utilizzare gli oggetti forniti dal modello a oggetti di Excel. Il modello a oggetti è una grande gerarchia di tutti gli oggetti utilizzati in VBA e il modello Application si riferisce e contiene i suoi oggetti di programmazione che sono legati gli uni agli altri in una gerarchia. L'intera applicazione Excel è rappresentata dall'oggetto Application che è in cima alla gerarchia di oggetti e scendendo verso il basso è possibile accedere agli oggetti per la cartella di lavoro, i fogli di lavoro e gli Intervalli di celle. Gli oggetti di Excel sono accessibili attraverso oggetti '*padri*', il foglio di lavoro è il genitore dell'oggetto Range, la cartella di lavoro è l'oggetto padre del foglio di lavoro, e l'oggetto Application è il padre dell'oggetto cartella di lavoro.

Brevemente si può dire che l'oggetto Application è la classe principale nel modello a oggetti e ogni volta che si apre Excel, viene istanziato un nuovo oggetto Application. La classe Application possiede metodi e proprietà come:

- *ActiveWorkBook*: proprietà che restituisce la cartella di lavoro attiva nell'applicazione
- *ActiveSheet*: proprietà che restituisce il foglio attivo nella cartella di lavoro attiva dell'applicazione
- *ActiveCell*: proprietà che restituisce la cella attiva nel foglio di lavoro attivo, della cartella di lavoro attiva dell'applicazione

Nel codice VBA, sia le espressioni *Application.ActiveWorkbook.Name* e *ActiveWorkbook.Name* avranno lo stesso effetto di restituire il nome della cartella di lavoro attiva. Tuttavia, ci sono alcuni casi in cui è richiesta la qualificazione dell'oggetto Application per essere utilizzato, vale a dire quando si utilizzano proprietà e metodi che riguardano l'aspetto della finestra di Excel, o che riguardano come l'applicazione Excel si deve comportare. Vediamo alcuni casi in cui il qualificatore Application deve essere utilizzato per proprietà o metodi dell'oggetto Application:

Proprietà Application.WindowState

La proprietà *WindowState* imposta lo stato della finestra dell'applicazione e le opzioni sono *xlNormal*, *xlMaximized* (imposta la finestra attiva alla dimensione massima disponibile purché non sia già ingrandita) e *xlMinimized*. Si noti che queste proprietà non possono essere impostate se la finestra è ingrandita, e le proprietà sono di sola lettura se la finestra è minimizzata

```
Sub app_winstat ()
  Application.WindowState = xlNormal
  Application.Height = 350
  Application.Width = 450
End Sub
```

Proprietà Application.DisplayFullScreen

Utilizza un valore Booleano e impostata su True, la finestra dell'applicazione è ingrandita alla dimensione massima e la barra del titolo dell'applicazione viene nascosta, sintassi:
Application.DisplayFullScreen = True

Proprietà Application.DisplayFormulaBar

Utilizza un valore Booleano e mostra o nasconde la barra della formula quando è impostato su, rispettivamente, True o False, sintassi: *Application.DisplayFormulaBar = False*

Proprietà Application.Calculation

Restituisce o imposta la modalità di calcolo e dispone di 3 impostazioni:

- *xlCalculationAutomatic* - (Default) ricalcola automaticamente i dati che vengono immessi nelle celle
- *xlCalculationSemiautomatic* - Ricalcolo automatico da Excel ad eccezione di tabelle di dati
- *xlCalculationManual* - Il calcolo viene fatto solo quando richiesto dall'utente facendo clic su 'Calcola ora' o premendo F9.

Esempio: *Application.Calculation = xlCalculationManual*

Proprietà Application.EditDirectlyInCell

Utilizza un valore booleano e consente o non consente la modifica direttamente nelle celle quando è impostata, rispettivamente a True o False, sintassi: *Application.EditDirectlyInCell = False*

Proprietà Application.ScreenUpdating

Utilizza un valore booleano ed è usata quando non si vuole vedere lo schermo seguire le azioni della routine VBA che si sta eseguendo. Se la proprietà *ScreenUpdating* è impostata su False, l'aggiornamento dello schermo non è visibile, e non sarà in grado di visualizzare ciò che il codice fa, rendendo più veloce l'esecuzione. Quando si disattiva l'aggiornamento dello schermo nelle procedure VBA al termine delle quali si deve impostare la proprietà a True, sintassi: *Application.ScreenUpdating = False*

Proprietà Application.DisplayAlerts

Utilizza un valore booleano e durante l'esecuzione di una macro, alcuni avvisi vengono visualizzati da Excel per confermare un'azione durante l'eliminazione di un foglio di lavoro. L'impostazione di questa proprietà su False non visualizzerà alcuna richiesta o avviso e in questo caso si riceverà una risposta predefinita da Excel. Questa proprietà deve essere impostata con il valore predefinito True dopo la procedura termina, sintassi: *Application.DisplayAlerts = False*

Proprietà Application.DefaultFilePath

Si usa questa proprietà per ottenere e impostare il percorso predefinito utilizzato da Microsoft Office Excel per caricare e salvare i file, sintassi: *Application.DefaultFilePath = "C:\Documenti\Excel"*

Metodo Application.Quit

Si utilizza questo metodo per chiudere l'applicazione Excel. Si noti che dopo la chiusura della cartella di lavoro, la finestra di Excel rimane aperta. Per uscire da Excel si può utilizzare il metodo Quit come illustrato di seguito. sintassi: *ThisWorkbook.Close SaveChanges: = True*

Chiude Excel e poi chiude ThisWorkbook dopo il salvataggio (il metodo Quit non termina Excel)
Application.Quit
ThisWorkbook.Close SaveChanges: = True

Chiude ThisWorkbook, ma non chiude Excel perché viene chiuso con ThisWorkbook senza leggere la linea Application.Quit:
ThisWorkbook.Close SaveChanges: = True
Application.Quit

Metodo Application.OnTime

Questo metodo viene utilizzato in VBA per eseguire automaticamente una procedura ad intervalli periodici o in un momento specifico della giornata. Nel seguente esempio, *RunTime* è una variabile pubblica di tipo Date, che imposta l'intervallo di tempo e la macro denominata MacroAutoRun verrà eseguito automaticamente, a intervalli di tempo programmato di tre secondi, con il metodo *OnTime,* sintassi: *Application.OnTime RunTime, "MacroAutoRun"*

Esempio di utilizzo del metodo Application.OnTime: Questa procedura utilizza il metodo OnTime col valore di incremento reperito in una cella a intervalli di tempo specifici, e Arresta la procedura dopo averla eseguita per un determinato numero di volte. Si Imposta l'intervallo di tempo 3 secondi, nel quale la procedura verrà eseguita

```
RunTime Public As Date
Dim count As Integer

Sub MacroAutoRun ()
RunTime = + TimeValue ("00:00:03") Now
Application.OnTime RunTime, "MacroAutoRun", True
 'si incrementa il valore nella cella A1 del foglio di lavoro di 5 unità, ogni volta che la macro si ripete
Cells (1, 1) Valore = Cells (1, 1).Value + 5
count = count + 1
'Interrompere la procedura dopo averla eseguita per 5 volte
If count = 5 Then
Application.OnTime RunTime, "MacroAutoRun", False
count = 0
End If
End Sub
```

Metodo Application.ActivateMicrosoftApp

Questo metodo attiva un'applicazione Microsoft già in esecuzione oppure crea una nuova istanza dell'applicazione nel caso in cui l'applicazione non è già in esecuzione. Qui di seguito i codici per avviare e attivare, rispettivamente, Word, Access e Power Point:

- *Application.ActivateMicrosoftApp xlMicrosoftWord*
- *Application.ActivateMicrosoftApp xlMicrosoftAccess*
- *Application.ActivateMicrosoftApp xlMicrosoftPowerPoint*

Metodo Application.GetOpenFilename

Il metodo *GetOpenFilename* ottiene il nome del file da aprire dall'utente, visualizzando la finestra di dialogo standard Apri. Il file non viene effettivamente aperto, ma restituisce il percorso completo e il nome del file selezionato. La sintassi è la seguente: *ApplicationObject.GetOpenFilename (FileFilter, FilterIndex, titolo, ButtonText, MultiSelect)*

Tutti gli argomenti sono opzionali, ma è necessario specificare l'oggetto Application.
L'argomento *FileFilter* è un valore stringa che specifica i criteri di filtro per il tipo di file che verranno visualizzati nella directory da cui l'utente ottiene il nome del file. Ci sono 4 filtri specificati in FileFilter ed è possibile specificare uno qualsiasi di questi criteri predefiniti utilizzando i numeri di indice da 1 a 4

Si può utilizzare l'argomento *Titolo* per specificare il titolo della finestra di dialogo. Il titolo predefinito è "Open" se non specificato. L'argomento *ButtonText* è applicabile solo per i computer che eseguono Excel per Macintosh (Mac) e impostando l'argomento *MultiSelect* a True si consente la selezione multipla dei file. L'impostazione predefinita è False. E' possibile utilizzare FileFilter per i file di Excel xlsm, specificando le estensioni in coppie e utilizzando caratteri jolly *"Excel Files (*. Xlsm), *. Xlsm"*

Si può usare FileFilter per estensioni xls, xlsx e XLSM, usando il punto e virgola e i caratteri jolly *"Excel Files (* xls,. *. Xlsx,. * Xlsm), * xls, *. Xlsx,. *. Xlsm"* Utilizzando FileFilter per estensioni XLSM e file di testo txt elencandoli separatamente nella lista dei tipi di file *"Excel Files (*. Xlsm), *. Xlsm, File di testo (*. Txt), *. Txt"*
Se l'argomento FileFilter viene omesso, per impostazione predefinita assume la sintassi: tutti i file *"Tutti i file (*. *), *. *"*

Esempio: Selezionare e aprire un singolo file, immettere il salvare e chiudere la cartella di lavoro, utilizzando i metodi Application.GetOpenFilename e Workbooks.Open

```
Sub GetOpenFilename1 ()
  Dim nome_file As Variant
  Dim wkbk As Workbook
  Dim str_cartella As String
  Dim cor_cartella As String
cor_cartella = CurDir
str_cartella = "C:\Test"
ChDrive str_cartella 'impostata l'unità corrente
ChDir str_cartella
nome_file = Application.GetOpennome_file(FileFilter:="Excel .. Files (* xlsm), * xlsm ",
Title:="Seleziona un file ")
If nome_file = False Then
MsgBox "Selezionare un file per continuare"
Exit Sub
Else
Workbooks.Open (nome_file)
Set wkbk = ActiveWorkbook
Sheets("Foglio1").Select
wkbk.ActiveSheet.Range("A1") = "Ciao"
 wkbk.Save
wkbk.Close
End If
ChDrive cor_cartella
ChDir cor_cartella
End Sub
```

Esempio: Selezionare e aprire più file, utilizzando i metodi Application.GetOpenFilename e Workbooks.Open

```
Sub GetOpenFilename2 ()
 Dim nome_file As Variant
 Dim i As Integer
 Dim iFiles As Integer
'impostando MultiSelect a True il metodo GetOpenFilename consente la selezione di file multipli
nome_file = Application.GetOpenFilename(filefilter:="Excel Files(*.xls;*.xlsx;*.xlsm),
*.xls;*.xlsx;*.xlsm", Title:="Seleziona file", MultiSelect:=True)
'Verifica se l'utente fa clic sul pulsante Annulla
If IsArray(nome_file) = False Then
MsgBox "Seleziona il file per continuare"
Exit Sub
Else
iFiles = UBound(nome_file) - LBound(nome_file) + 1 'Determinare il numero di file da aprire
For i = 1 To iFiles
Workbooks.Open nome_file(i) 'Metodo Workbooks.Open apre una cartella di lavoro
Next i
End If
End Sub
```

Esempio: Aprire un file Excel esistente, aggiungere una nuova cartella nella stessa posizione e salvarlo come un file xlsm, poi copiare un foglio dalla cartella di lavoro esistente alla cartella di lavoro appena aggiunta, quindi chiudere entrambe le cartelle di lavoro.

```
Sub GetOpenFilename3()
Dim fileName As Variant
Dim str_cart As String
Dim wb_dest As Workbook
Dim wb_orig As Workbook
str_cart = "C:\Test" 'impostare un percorso da cui selezionare i file nella finestra di dialogo Apri
ChDrive str_cart 'impostare l'unità
ChDir str_cart 'impostare la directory corrente
fileName = Application.GetOpenFilename(FileFilter:="Excel .. Files (* xlsm), * xlsm ",
Title:="Seleziona un file ")
If fileName = False Then 'Se l'utente fa clic sul pulsante Annulla viene visualizzato un avviso a video
MsgBox "Si prega di selezionare un file per continuare"
Exit Sub
Else
Set wb_orig = Workbooks.Open(fileName) 'apre una cartella di lavoro
End If
Workbooks.Add 'Aggiungere una nuova cartella di lavoro, che è la cartella di lavoro di destinazione
'salva la cartella di lavoro di destinazione con un nuovo nome e il tipo di file xlsm
ActiveWorkbook.SaveAs fileName:="newWorkbook.xlsm", FileFormat:=52
Set wb_dest = ActiveWorkbook
'copia il foglio2 dalla cartella di origine alla nuova cartella di destinazione come ultimo foglio
wb_orig.Worksheets("Foglio2").Copy After:=wb_dest.Sheets(Sheets.Count)
'chiude sia le cartelle di lavoro, dopo aver salvato la cartella di destinazione
wb_orig.Close
wb_dest.Close SaveChanges:=True
End Sub
```

Proprietà e metodi dell'oggetto Application

Tali proprietà e metodi il cui utilizzo **NON** richiede di specificare il qualificatore dell'oggetto Application sono considerati *globali*. È possibile visualizzare queste proprietà e metodi globali nel Visualizzatore oggetti in VBE dal menu **Visualizza - Visualizzatore oggetti**, o premendo F2 e scegliendo Excel dall'elenco a discesa delle librerie nel riquadro superiore e quindi scegliendo *GLOBALS* che appare in alto nella casella Classi. I Casi in cui non è richiesta l'utilizzazione del qualificatore dell'applicazione sono:

Proprietà ActiveCell

La proprietà *ActiveCell*, applicata ad un oggetto Application, restituisce la cella attiva (oggetto Range) del foglio di lavoro visualizzato nella finestra attiva. È inoltre possibile applicare questa proprietà per un oggetto *Window* specificando la finestra per cercare la cella attiva. Si noti che, in assenza di un foglio di lavoro visualizzato nella finestra, la proprietà avrà esito negativo. Qualsiasi dei seguenti codici possono essere utilizzati in alternativa (si noti che Value è la proprietà predefinita di un oggetto Range)

MsgBox "il valore della cella attiva è:" & Application.ActiveCell
MsgBox "il valore della cella attiva è:" & ActiveCell
MsgBox "il valore della cella attiva è:" & ActiveWindow.ActiveCell
MsgBox "il valore della cella attiva è:" & ActiveCell.Value

Proprietà ActiveWindow

Questa proprietà restituisce la finestra attiva. La finestra attiva è la finestra attualmente selezionata. Il codice per visualizzare il nome del'[i]ActiveWindow[i] che appare nella barra del titolo è il seguente:

MsgBox "La finestra attiva è:" & Application.ActiveWindow.Caption
MsgBox "La finestra attiva è:" & ActiveWindow.Caption

Proprietà ActiveWorkbook

Questa proprietà restituisce la cartella di lavoro attiva, cioè la cartella di lavoro nella finestra attiva

MsgBox "il nome della cartella attiva è" & Application.ActiveWorkbook.Name
MsgBox "il nome della cartella attiva è" & ActiveWorkbook.Name

Proprietà ThisWorkbook

Questa proprietà viene utilizzata solo dall'interno dell'applicazione Excel e restituisce la cartella di lavoro in cui il codice viene eseguito al momento.

MsgBox "Il nome di questa cartella di lavoro è" & Application.ThisWorkbook.Name
MsgBox "Il nome di questa cartella di lavoro è" & ThisWorkbook.Name

Si noti che sebbene il più delle volte ActiveWorkbook è la stessa ThisWorkbook, ma potrebbe non essere sempre così. La cartella di lavoro attiva può essere diversa da quella di lavoro in cui viene eseguito il codice, come illustrato dal seguente esempio.

```
Sub ActiveWorkbook_ThisWorkbook ()
MsgBox "il nome della cartella Attiva è" & ActiveWorkbook.Name
MsgBox "Il nome di questa cartella di lavoro è" & ThisWorkbook.Name
Workbooks ("Book2. xlsm "). Activate
MsgBox "il nome della cartella Attiva è" & ActiveWorkbook.Name
MsgBox "Il nome di questa cartella di lavoro è" & ThisWorkbook.Name
Workbooks ("Book1.xlsm"). Activate
MsgBox "il nome della cartella Attiva è" & ActiveWorkbook.Name
MsgBox "Il nome di questa cartella di lavoro è" & ThisWorkbook.Name
End Sub
```

Proprietà ActiveSheet

La proprietà *ActiveSheet*, applicata ad un oggetto Application, restituisce il foglio attivo nella cartella di lavoro attiva. È inoltre possibile applicare questa proprietà per una cartella di lavoro o finestra specificando la cartella di lavoro o la finestra per cercare il foglio attivo. I seguenti codici mostrano il foglio attivo nella cartella di lavoro attiva

Msgbox "il nome del foglio attivo è" & Application.ActiveSheet.Name
MsgBox "il nome del foglio attivo è" & Application.ActiveWorkbook.ActiveSheet.Name
Msgbox "il nome del foglio attivo è" & ActiveSheet.Name
MsgBox "il nome del foglio attivo è" & ActiveWorkbook.ActiveSheet.Name

Mentre il seguente codice restituisce il foglio attivo nella cartella di lavoro specificata (denominata "Test_vba.xlsm"):

MsgBox "il nome del foglio attivo è" & Application.Workbooks ("Test_vba.xlsm"). ActiveSheet.Name

Proprietà ActiveChart

La proprietà *ActiveChart*, applicata ad un oggetto Application, restituisce il grafico attivo nella cartella di lavoro attiva. Nella cartella di lavoro è possibile avere fogli grafici separati e definisce grafici come fogli di lavoro o grafici incorporati che comprende il grafico come un oggetto all'interno di un foglio di lavoro. Un foglio grafico o un grafico incorporato è attivo se è stato selezionato o se è attivato con il metodo Activate. È inoltre possibile applicare la proprietà ActiveChart di una cartella di lavoro o un oggetto Window specificando la cartella di lavoro o la finestra per cercare il grafico attivo. I seguenti codici mostrano il grafico attivo nella cartella di lavoro attiva

MsgBox "Il nome del grafico attivo è:" & Application.ActiveChart.Name
MsgBox " Il nome del grafico attivo è:" & Application.ActiveWorkbook.ActiveChart.Name
MsgBox " Il nome del grafico attivo è:" & ActiveChart.Name
MsgBox " Il nome del grafico attivo è:" & ActiveWorkbook.ActiveChart.Name

Il seguente codice restituisce il grafico attivo nella cartella di lavoro specificata (denominata " Test_vba.xlsm ")

MsgBox " Il nome del grafico attivo è:" & Application.Workbooks ("Test_vba.xlsm ") ActiveChart.Name

Proprietà Application.ActivePrinter

Questa proprietà imposta il nome della stampante attiva. Per Impostare una stampante HP LaserJet 0011 su LPT1 come stampante attiva

Application.ActivePrinter = "HP LaserJet 0011 su NE02: "
ActivePrinter = "HP LaserJet 0011 su NE02: "

Per impostare la stampante Adobe PDF sul Ne04 come stampante attiva

Application.ActivePrinter = "Adobe PDF su Ne04: "
ActivePrinter = "Adobe PDF su Ne04: "

Proprietà Selection

La proprietà *Selection* se applicata a un oggetto Application, restituisce l'oggetto che viene selezionato nella finestra attiva. L'oggetto selezionato potrebbe essere un oggetto Range o una singola cella del foglio di lavoro attivo, un oggetto ChartArea nel foglio di lavoro attivo, e così via. È inoltre possibile applicare la proprietà di selezione un oggetto finestra in cui la proprietà restituirà l'oggetto selezionato nella finestra specificata. Presumendo che la selezione corrente sia un oggetto Range nel foglio di lavoro attivo.

Per cancellare il contenuto delle celle selezionate nel foglio di lavoro attivo, si può utilizzare uno dei due metodi sotto descritti con o senza il qualificatore di oggetto Application

Application.Selection.Clear
Selection.Clear

Esempio: Selezionare un intervallo di celle per ottenere il tipo di oggetto della selezione e utilizzare la selezione per cambiare colore e grandezza carattere.

```
Sub SelectionProperty ()
Range ("A1: C3"). Select
'ottenere il tipo di oggetto della selezione - restituisce "Range"
MsgBox TypeName (Selection)
'cambiare il colore del carattere del campo selezionato
Selection.Font.Color = vbRed
'cambiare la dimensione del carattere dell'intervallo selezionato
Selection.Font.Size = 14
End Sub
```

Proprietà Fogli

L'oggetto *Sheets* si riferisce a tutti i fogli contenuti in una cartella di lavoro, che comprende fogli grafici e fogli di lavoro. Le Schede della struttura, applicate ad un oggetto Application, restituisce un insieme Sheets nella cartella di lavoro attiva. È inoltre possibile applicare questa proprietà per una cartella di lavoro specificando la cartella di lavoro che restituirà una raccolta di fogli di lavoro specificato. Qui di seguito viene illustrato come contare il numero di fogli della cartella di lavoro attiva e restituire il nome di ciascun foglio.

```
Sub SheetsCollection ()
Dim n As Integer
'numero dei fogli nella cartella di lavoro attiva
MsgBox " Numero di fogli nella cartella di lavoro attiva sono:" & Application.Sheets.Count
MsgBox " Numero di fogli nella cartella di lavoro attiva sono:" & Sheets.Count
'Nome dei fogli in base al valore dell'indice
For n = 1 To Sheets.Count
MsgBox Sheets(n). Name
Next n
'Nome dei fogli scorrendoli tutti nella cartella di lavoro
For Each Sheet In Sheets
MsgBox Sheet.Name
Next
End Sub
```

Proprietà Intervallo

La proprietà *Range* restituisce un oggetto Range (singola cella o intervallo di celle). È possibile utilizzare la sintassi: *Range (cell1)*. Questo può essere solo un riferimento di tipo A1, ed è possibile utilizzare un operatore di intervallo o l'operatore di unione (es. virgola), oppure può essere un intervallo denominato. È inoltre possibile utilizzare la sintassi: *Range (cell1, cell2)* - dove cell1 e cell2 sono oggetti che rappresentano l'intervallo di celle contigue che specificano le celle di inizio e fine. Per applicare la proprietà Range di un oggetto Application, è possibile omettere il qualificatore di oggetto. Range (cell1) o utilizzare *Application.Range (cell1)*, entrambi restituiranno l'oggetto Range in ActiveSheet. Per applicare la proprietà Range di un oggetto cartella di lavoro o un oggetto Range, specificare il rispettivo qualificatore oggetto come illustrato di seguito.

Proprietà Range applicabili all'oggetto Application

Qualsiasi dei seguenti codici inseriranno il valore 10 nella cella A1 del foglio attivo (con o senza usare il qualificatore di oggetto Application o digitando ActiveSheet)

Application.ActiveSheet.Range ("A1"). Value = 10
ActiveSheet.Range ("A1"). Value = 10
Application.Range ("A1"). Value = 10
Range ("A1"). Value = 10

Il seguente codice inserirà il valore 10 nelle celle A1, A2, A3 e A4 del foglio attivo. La seconda espressione utilizza la sintassi: Range (cell1, cell2):

Range ("A1: A4") Valore = 10
Range ("A1", "A4"). Value = 10

Il seguente codice inserirà il valore 10 nelle celle A1, B2 e C4 del foglio attivo (si noti che specificando "Value" dopo il Range non è necessario perché è la proprietà predefinita dell'oggetto Range)

Range ("A1, B2, C4") = 10

Il seguente codice inserirà il valore 10 nell'intervallo denominato "pippo" del foglio attivo, vale a dire che è possibile denominare l'intervallo A1 come "pippo" e utilizzare il seguente codice:

Range ("pippo") = 10

Proprietà Range applicabili all'oggetto Worksheet

Il seguente codice inserirà il testo "Carlo" nella cella A1 del foglio di lavoro denominato "Foglio1": *Worksheets ("Foglio1"). Range ("A1") = "Carlo"*

Proprietà Range applicabili all'oggetto Range

Quando la proprietà Range viene applicata a un oggetto Range, la proprietà diventa relativa all'oggetto Range come illustrato di seguito:

```
Sub RangeProperty ()
'Selezionare un intervallo nel foglio attivo
Range("C5:E8").Select
'inserisce il valore 10 in C5
Selection.Range("A1") = 10
'inserisce il valore 11 in C6
Selection. Range ("A2") = 11
'inserisce il valore 20 in D5
Selection.Range ("B1") = 20
'inserisce il valore 20 in D6
Selection.Range ("B2") = 21
End Sub
```

Metodo Calcola

Il metodo Calcola, se applicato a un oggetto Application, calcola tutte le cartelle di lavoro aperte. È inoltre possibile applicare questo metodo a un oggetto foglio di lavoro specificando il foglio di lavoro in una cartella di lavoro; oppure è possibile applicare questo metodo a un intervallo specificando una cella o un intervallo di celle in un foglio di lavoro.

Metodo Calcola applicabile all'oggetto Application
Utilizzare una delle seguenti righe di codice per calcolare tutte le cartelle di lavoro aperte:
Application.Calculate
Calculate

Metodo Calcola applicabile all'oggetto Worksheet
Utilizzare una delle seguenti righe di codice per calcolare un foglio di lavoro specifico ("Foglio1"):
Application.Worksheets ("Foglio1"). Calculate
Worksheets ("Foglio1"). Calculate

Metodo Calcola applicabile all'oggetto Range
Utilizzare una delle seguenti righe di codice per calcolare l'intervallo specificato (celle A5, A6 e A7) in un foglio di lavoro:
Application.Worksheets ("Sheet1") Range ("A5: A7"). Calculate.
Worksheets ("Foglio1") Range ("A5: A7"). Calculate

La seguente riga di codice calcola l'intera colonna (colonna A) in un foglio di lavoro:
Worksheets ("Foglio1"). Columns (1). Calculate

La seguente riga di codice calcola le celle specifiche A5, A6, B7 e B20 in un foglio di lavoro:
Worksheets ("Foglio1"). Range ("A5, A6, B7, B20"). Calculate

Utilizzare la seguente riga di codice per calcolare l'intervallo specificato (celle A5, A6 e A7) nel foglio di lavoro attivo: *Range ("A5: A7"). Calculate*

Velocizzare il codice VBA disattivando gli aggiornamenti dello schermo e i calcoli automatici
In Excel, la modalità di calcolo predefinita è la Modalità Automatica. sintassi: (*Application.Calculation = xlCalculationAutomatic*), in cui viene calcolata automaticamente ogni cella quando si entra. Quando Excel è in modalità manuale: (*Application.Calculation = xlCalculationManual*)

il calcolo viene effettuato solo quando richiesto dall'utente facendo clic su "*Calcola Ora*" o premendo F9 oppure cambiando la modalità di calcolo automatico. In modalità automatica, per ciascun nuovo valore immesso da una macro, Excel ricalcolerà tutte le celle che verranno coinvolte dal nuovo valore rallentando così l'esecuzione del codice VBA. Questo può rallentare notevolmente il funzionamento del codice vba, soprattutto nel caso di grandi macro con moli di calcolo significativi. Per accelerare una macro e rendere l'esecuzione più veloce ed efficiente, è tipico degli sviluppatori disattivare il calcolo automatico all'inizio della macro, ricalcolare il foglio di lavoro specifico o l'intervallo di celle utilizzando il metodo Calculate all'interno della macro e poi riportare il calcolo in automatico alla fine del codice.

L'esempio seguente disattiva gli aggiornamenti dello schermo e i calcoli automatici e utilizza il metodo Calculate, durante l'esecuzione del codice

```
Sub CalculateMethod ()
Application.ScreenUpdating = False 'disattivare Aggiornamenti schermo e calcoli automatici
Application.Calculation = xlCalculationManual
..... altro codice
Application.Calculate 'Utilizzare il metodo Calculate per calcolare tutte le cartelle di lavoro aperte
Application.ScreenUpdating = True 'attivare gli aggiornamenti dello schermo e calcoli automatici
Application.Calculation = xlCalculationAutomatic
End Sub
```

Prendere decisioni – Ciclo If e Select Case

Finora abbiamo visto delle procedure che sono in grado di portare a termine i compiti assegnati, ma però non sono in grado di prendere delle decisioni che permettano di eseguire diverse azioni in circostanze differenti, operazione che risultano necessarie in molte situazioni. A volte è necessario che sia la procedura stessa ad offrire la possibilità di poter scegliere quale azione intraprendere al verificarsi di un determinato evento, per esempio possiamo scrivere una procedura che controlli una colonna in un foglio di lavoro per verificare se tutti i numeri sono compresi tra 1 e 10, inoltre la procedura potrebbe poi esaminare ogni elemento della colonna separatamente ed eseguire azioni particolari se incontra un elemento non compreso nell'intervallo specificato.

Il Ciclo IF

Poichè le istruzioni per l'esecuzione di scelte modificano il flusso di esecuzione del programma, vengono spesso chiamate istruzioni di *controllo di flusso* o di *controllo di programma*, ma sono più note tecnicamente agli addetti ai lavori come istruzioni *condizionali* e *incondizionali*.
Un'istruzione *condizionale* è una struttura per l'esecuzione di scelte, che sceglie un blocco di istruzioni del codice del programma basandosi su una condizione o su un gruppo di condizioni predefinite, mentre un'istruzione *incondizionale* è un'istruzione che modifica semplicemente il flusso di esecuzione della procedura senza dipendere da nessuna condizione specifica. Vediamo in dettaglio questo concetto. Per eseguire un'istruzione condizionale usiamo la funzione **If** ed è rappresentata in due modi

If Condizione Then
Istruzioni
End If

Oppure a riga singola

If Condizione Then Istruzioni

Quando VBA incontra un'istruzione condizionale come If ....Then, prima valuta l'espressione logica che descrive le condizioni, in base alle quali deve essere eseguita una particolare azione, se l'espressione è *True* (cioè vera) le condizioni predefinite sono state soddisfatte e vengono eseguite le istruzioni indicate, mentre ***End If*** indica la fine del ciclo decisionale. Vediamo questa procedura con degli esempi

```
Sub prova()
 Dim var1 As Integer
 var1 = "1"
 If var1 = "1" Then MsgBox ("Bravi")
End Sub
```

In pratica la funzione fa questa valutazione: *se la variabile var1 è uguale a "1"*, [Valutazione delle condizioni], e la valutazione è *True* (cioè è vera), allora fai apparire un messaggio con la scritta *Bravi*, (Esegui l'istruzione), possiamo anche specificare più di un'azione da intraprendere in base alle condizioni usando il seguente enunciato

If Condizione Then
Istruzioni
Else
Istruzioni per Else
End If

In pratica da quanto sopra esposto aggiungiamo altre istruzioni nel caso che la condizione non venga soddisfatta, lo possiamo capire meglio con questo esempio

```
Sub prova1()
Dim var1 As Integer
var1 = "2"
If var1 = "1" Then
 MsgBox ("Bravi")
Else
 MsgBox ("Condizione non soddisfatta")
End If
End Sub
```

Oppure usando l'enunciato a riga singola

```
Sub prova()
Dim var1 As Integer
var1 = "2"
If var1 = "1" Then MsgBox ("Bravi") Else MsgBox ("Condizione non soddisfatta")
End Sub
```

Finora abbiamo visto delle istruzioni condizionali capaci di scegliere un singolo blocco di istruzioni alternativo per l'esecuzione della procedura, in molti casi però abbiamo bisogno di fare delle scelte più complesse scegliendo fra tre o quattro o più blocchi di istruzioni da eseguire, possiamo in questo caso inserire delle istruzioni *If ... Then* o *If ... Then .. Else* all'interno di altre istruzioni If ... Then o If ... Then .. Else, questa operazione si chiama **"Nidificare"** le istruzioni (nidificare significa mettere un tipo di struttura di controllo del flusso all'interno di un'altra), per usare questa sintassi è meglio usare il formato a blocchi, per una maggiore chiarezza e semplicità di lettura, l'enunciato è espresso in questa forma

If Condizione Then
Istruzioni
Else
If Condizione1 Then
Istruzioni1
Else
Istruzioni 2
End If
End If

Vediamo con un esempio come nidificare più istruzioni ed usiamo anche la funzione InputBox che abbiamo visto all'inizio

```
Sub nidifica()
Dim var1
var1 = InputBox(prompt:="Inserisci a quanti gradi metti il termostato del riscaldamento: ",
Title:="Misura la temperatura di casa")
If Pianeta > 20 Then
 MsgBox "Troppo caldo, vedrai che bolletta"
Else
  If var1 > 18 Then
  MsgBox "Temperatura giusta"
  Else
  MsgBox "Temperatura troppo bassa, ti prendi un raffreddore"
  End If
End If
End Sub
```

Eseguendo questa macro e inserendo il valore 30 otteniamo questo:

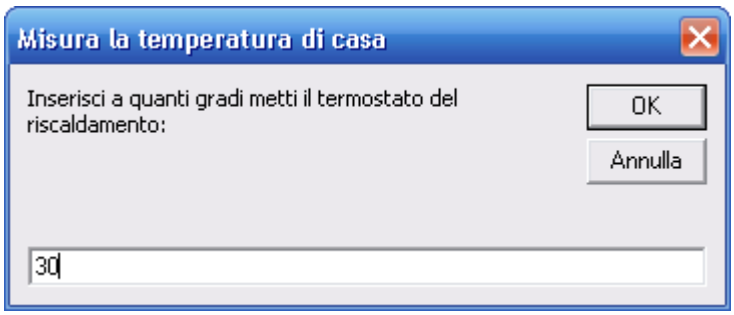


Fig. 1

Fig. 2

Se invece inseriamo il valore 19 otteniamo questi messaggi

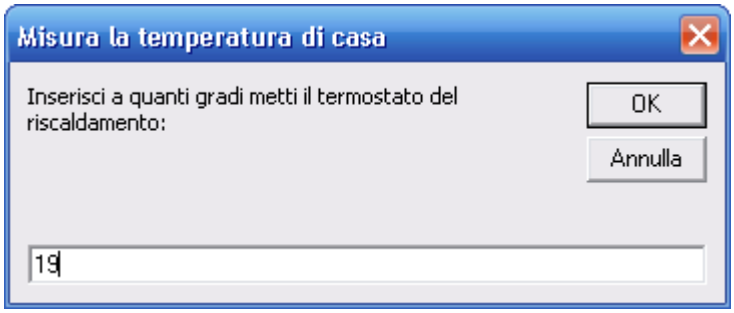


Fig. 3

Fig. 4

Inserendo invece il valore 17 ci viene mostrato questo avviso

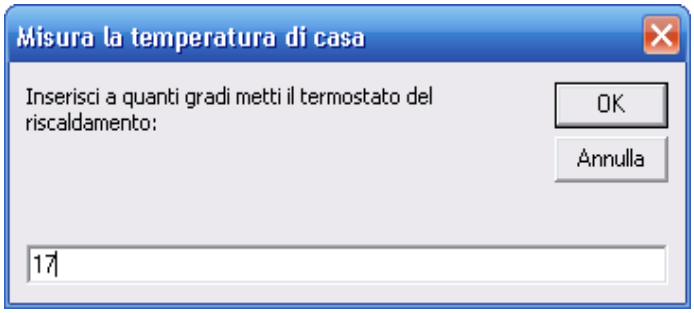


Fig. 5

Fig. 6

Vediamo nel dettaglio casa abbiamo fatto, per comprendere meglio la nidificazione usiamo anche i colori, all'inizio abbiamo dichiarato la variabile var1 ed abbiamo omesso di specificare il tipo di dati (ricordate che nella lezione 3 abbiamo detto che non dichiarando il tipo di dati la variabile assumeva per default il tipo Variant, in questo esempio è di scarsa importanza il tipo di dati), poi tramite la funzione InputBox abbiamo richiesto un dato dall'utente e di seguito abbiamo elencato le condizioni.

La prima *If var1 > 20 Then* confronta il valore della variabile var1 e avendole assegnato il valore 30 tramite InputBox viene soddisfatta la prima condizione che abbiamo posto, cioè, se var1 è maggiore di 20, porta a video il messaggio *Troppo caldo, vedrai che bolletta* a questo punto che la condizione è stata verificata e soddisfatta l'esecuzione continua dopo la parola chiave *End If*, *in questo caso segue la procedura di colore blu* e ci riporta a fine codice.

Se nell'InputBox inseriamo un valore diverso (ad esempio 20) la prima condizione *If var1 > 20 Then* **non** viene soddisfatta ed allora passiamo alle istruzioni *Else* dove, in questo blocco di istruzioni, abbiamo posto 2 condizioni, la prima che viene verificata è *If var1 > 18 Then* (Se var1 è maggiore di 18), porta a video il messaggio *Temperatura giusta*, nel nostro caso avendo inserito il valore 20 viene soddisfatta questa condizione in quanto è maggiore di 18. Nello stesso blocco *Else* abbiamo anche posto un'altra condizione senza nessun valore, che significa, semplicemente che se il valore che introduciamo con InputBox non soddisfa la condizione If var1 > 18 Then allora il flusso del programma esegue le istruzioni di questa istruzione, infatti se inseriamo il valore 15 ci viene riportato a video il messaggio "Temperatura troppo bassa, ti prendi un raffreddore".

Ma perchè queste scelte avvengono così? perchè le istruzioni *Else* sono contenute completamente all'interno dell'istruzione più esterna, e quando viene verificato il valore della variabile Var1 viene mandato in esecuzione il blocco di codice quando il valore della variabile è *True* (vera) ne consegue che inserendo il valore 30 Var1 diventa *True* alla prima condizione ed esegue il flusso di *codice di colore blu*, invece se assegniamo a Var1 il valore 20 diventa *True* alla prima istruzione delle condizioni nidificate nel blocco *Else*, ogni altro valore assegnato [da 1 a 18] alla variabile Var1 viene eseguita l'ultima istruzione del blocco Else.

Possiamo semplificare il listato della nidificazione usando un'abbreviazione che si presenta con If ... Then .. ElseIf e la possiamo rappresentare con il seguente enunciato

If Condizione Then
Istruzioni
ElseIf Condizione1 Then
Istruzioni1
Else
Istruzioni 2
End If

Consideriamo questo enunciato come un'abbreviazione, il concetto sopra esposto non varia, la nostra macro diventa così

```
Sub nidifica_abbreviato()
Dim var1
var1 = InputBox(prompt:="Inserisci a quanti gradi metti il termostato del riscaldamento: ",
Title:="Misura la temperatura di casa")
If var1 > 20 Then
 MsgBox "Troppo caldo, vedrai che bolletta"
ElseIf var1 > 18 Then
 MsgBox "Temperatura giusta"
Else
 MsgBox "Temperatura troppo bassa, ti prendi un raffreddore"
End If
End Sub
```

Funzione Select Case

Le procedure che abbiamo visto finora sono moto utili con un numero ristretto di scelte da effettuare, ma presentano un problema quando ci sono molte condizioni da verificare, tale problema è di natura interpretativa in quanto diventa difficile leggere ed interpretare il listato del codice. VBA ci offre però un'istruzione condizionale da usare quando dobbiamo scegliere tra un gran numero di possibili scelte.

L'istruzione **Select Case** funziona allo stesso modo di più istruzioni **IF** indipendenti ma è più facile da eseguire ed interpretare, usando la parola chiave *Select Case* con più istruzioni *Case*, dove ogni istruzione Case verifica la presenza di una condizione e se saranno soddisfatte le condizioni, verrà eseguito il codice di un solo blocco Case, inoltre un blocco Case può contenere nessuna, una o più istruzioni, pertanto da questa breve introduzione possiamo affermare che in presenza di varie scelte da effettuare risulta un metodo molto utile e versatile oltre che facilmente leggibile. L'istruzione Select Case ha questa sintassi

Select Case espressione
Case elencoespressione1
Istruzioni1
Case elencoespressione2
Istruzioni2
etc...
Case elencoespressione n
[Case Else
IstruzioniElse]
End Select

Come già citato il concetto e l'utilizzo è uguale a più istruzioni IF, Select Case in più offre la possibilità di poter operare varie scelte e anche di utilizzare un operatore per specificare un intervallo di valori nell'elenco espressioni, tale operatore è **To** ed è espresso in questa forma: *espressione1 To espressione2*, per esempio possiamo specificare un intervallo di numeri da 1 a 10 in un elenco Case usando la seguente dicitura

Case 1 To 10

è inoltre possibile usare degli operatori di confronto per selezionare dei blocchi di istruzioni a seconda se espressione è maggiore, minore o uguale di espressione, la sintassi è la seguente:

Is operatore_di_confronto espressione, che semplificato prende questa forma: *Case Is < 10*

Vediamo ora con un esempio pratico di semplificare ulteriormente quanto esposto e trasformiamo la routine utilizzata con l'operatore *IF* con *Select Case*

```
Sub Selec_cast()
Dim var1
var1 = InputBox(prompt:="Inserisci a quanti gradi metti il termostato del riscaldamento: ",
Title:="Misura la temperatura di casa")
Select Case var1
  Case Is > 25
    MsgBox "Troppo caldo, vedrai che bolletta"
  Case 21 To 23
    MsgBox "Bello caldo, si stà bene"
  Case 17 To 20
    MsgBox "Temperatura giusta"
  Case > 15
    MsgBox "Raffredore assicurato"
  Case Else
    MsgBox "Troppo freddo si ghiaccia"
End Select
End Sub
```

Diamo una breve spiegazione a conclusione di questa lezione, sul listato appena esposto, avrete notato che è molto intuitivo, ma vediamo assieme come vengono interpretate le varie condizioni.

Abbiamo dichiarato una variabile [var1] ed abbiamo assegnato alla stessa un valore fornito dall'utente tramite la funzione InputBox, all'inizio del ciclo Select Case notiamo che è presente una sola variabile [var1], come abbiamo citato nella lezione sulle variabili il risultato di un'espressione contenente una singola variabile è il valore memorizzato nella variabile stessa, di conseguenza VBA confronta il valore memorizzato nella variabile Var1 con le condizioni specificate in ogni blocco *Case* dell'istruzione *Select Case*.

Innanzi tutto VBA controlla il valore della variabile var1 partendo dalla prima clausola Case, se assegniamo alla variabile il valore 28 viene subito soddisfatta la prima condizione Case Is > 25 *è maggiore di 25?* in questo caso si, la condizione diventa *True* e vengono eseguite le istruzioni di quel blocco Case, allo stesso modo se la variabile assume il valore 22, la prima condizione non viene soddisfatta [non è maggiore di 25] e VBA passa alla seconda [il valore è compreso tra 21 e 23?], Si, pertanto la condizione diventa *True* al secondo blocco Case ed esegue le relative istruzioni.
In buona sostanza possiamo mettere diverse condizioni, sia singole che intervalli,
nell'enunciato *Else* vengono invece inserite le istruzioni nel caso nessuna delle condizioni elencate venga soddisfatta, in questo caso vengono eseguite le istruzioni presenti nel blocco Else

Utilizzare la parola chiave To

Si può utilizzare la parola chiave To nell'espressione da valutare per specificare l'intervallo superiore e inferiore dei valori corrispondenti, come illustrato di seguito. Il valore a fianco della parola chiave To deve essere minore o uguale al valore a destra della parole chiave To.

```
Sub Test1 ()
Dim voti As Integer
voti = InputBox ("Inserisci Voto")
Select Case voti
Caso 70 To 100
MsgBox "Buono"
Caso 40 To 69
MsgBox "Medio"
Caso 0 To 39
MsgBox "Bocciato"
Case Else
MsgBox "Fuori Valutazione"
End Select
End Sub
```

Utilizzare la parola chiave Is

Per includere un operatore di confronto (=, <>, <,>, <= o> =) nell'espressione da valutare si utilizza la parola chiave Is, che viene inserita automaticamente prima di un operatore di confronto, se non espressamente incluso.

```
Sub Test2 ()
'se temperatura è uguale a 39,5, verrà restituito il messaggio "Moderatamente caldo"
Dim temp As Single
temp = 39.5
Select Case temp
Caso Is > = 40
MsgBox "Troppo Caldo"
Caso Is > = 25
MsgBox "Moderatamente caldo"
Caso Is > = 0
MsgBox "Troppo Freddo"
Caso Is <0
MsgBox "Molto Freddo"
End Select
End Sub
```

Utilizzare una virgola per separare più espressioni

Si possono specificati più espressioni o intervalli in ogni istruzione Case, separando ogni espressione con una virgola, che ha l'effetto dell'operatore OR. Più espressioni o intervalli possono essere specificati per stringhe di caratteri.

```
Sub Test3 ()
Dim var As Variant
var = "Ciao"
Select Case var
Case a, e, i, o, u
MsgBox "Vocali"
Cas2 2, 4, 6, 8
MsgBox "Numeri"
Caso 1, 3, 5, 7, 9, "Ciao"
MsgBox "Numeri o Ciao"
Case Else
MsgBox "Non Valutabile"
End Select
End Sub
```

Esempio: Confronto tra stringhe "mele" A "uve" determina un valore compreso tra "mele" e "uve" in ordine alfabetico, e utilizza il metodo di confronto di testo predefinito in Binary (che è case-sensitive), perché Istruzione Option Compare è non specificata

```
Sub Test4 ()
'Option Compare non è specificato e quindi il confronto testo sarà case-sensitive
Dim var As Variant, risultato As String
var = InputBox ("Inserisci dati")
Select Case var
Case 1 To 10, 11 To 20: risultato = "Il numero è compreso tra 1 e 20"
Case "mele" To "uva", "mango", 98, 99: risultato = "Testo tra mele e uva, o mango, oppure
tra i numeri 98 o 99"
Case Else: risultato = "Non Valutabile"
End Select
MsgBox risultato
End Sub
```

Impostazione Option Compare

È possibile confrontare i dati stringa utilizzando metodi di confronto tra stringhe in binario, testo o database. (quest'ultimo viene utilizzato solo con Microsoft Access). Option Compare Binary rende confronti di stringhe sulla base di un ordinamento binario (in Microsoft Windows, la pagina di codice determina il tipo di ordinamento - in cui ANSI 1252 è utilizzato per l'inglese e molte lingue europee), inoltre Option Compare Text rende i confronti di stringhe che non si basano su un ordinamento testuale case-sensitive

L'istruzione Option Compare (cioè Option Compare Binary o Option Compare Text) può essere utilizzato per impostare il metodo di confronto e deve essere utilizzato a livello di modulo, prima di qualsiasi procedura. Se l'istruzione Option Compare non è specificata, il metodo di confronto testo predefinito è Binary.

```
Option Compare Binary
Sub Compare1 ()
Dim str As String
str = InputBox("Inserisci il testo ")
Select Case str
Case "Mele" To "Uva"
MsgBox " Il testo è tra mele e uva "
Case Else
MsgBox "Non Valutabile"
End Select
End Sub

Option Compare Text
Sub Compare2 ()
Dim str As String
str = InputBox ("Inserisci il testo")

Select Case str
Case "mele" To "uve"
MsgBox "Il testo è tra mele e uva"
Case Else
MsgBox "Non Valutabile"
End Select
End Sub
```

Select Case Annidati

Il blocco di istruzioni Select Case può essere nidificato all'interno di ogni altro ciclo, come If ... Then ... Else e Loop, senza alcun limite. Quando Select Case è nidificato dentro l'altro, deve essere un blocco completo e terminare con la propria End Select , all'interno di una specifica Case o Case Else

```
Sub select1 ()
Dim rng As Range, int1 As Integer
Set rng = ActiveSheet.Range ("A1")
Select Case IsEmpty (rng)
Case True
MsgBox rng.Address & "è vuota"
Case Else
Select Case IsNumeric (rng)
Case True
MsgBox rng.Address e "ha un valore numerico"
Select Case rng.HasFormula
Case True
MsgBox rng.Address & "ha anche una formula"
End Select
Case Else
Int1 = Len (rng)
MsgBox rng.Address & "ha una lunghezza di testo di" & int1
End Select
End Select
End Sub
```

Esempio: Manipolazione del testo con le istruzioni condizionali nidificati

```
Funzione StringManipulation (str As String) As String
'Questo codice personalizza una stringa di testo come segue:
`1. rimuove i numeri da una stringa di testo;
'2. rimuove gli spazi iniziali e finali
'3. aggiunge uno spazio (se non presente) dopo ogni esclamativo, virgola, punto e interrogativo;
Dim iTxtLen As Integer, iStrLen As Integer, n As Integer, i As Integer, ansiCode As Integer
'Toglie i numeri
'Chr (48) chr (57) rappresentano i numeri da 0-9 in codici ASCII
For i = 48 To 57
'Rimuovere tutti i numeri dalla stringa di testo
str = Replace (str, Chr (i), "" )
Next i
'Toglie gli spazi con la funzione TRIM
str = Application.Trim (str)
 'Aggiunge uno spazio (se non presente) dopo ogni ! ; ? .
iTxtLen = Len (str)
For n = iTxtLen To 1 Step -1
'Chr spazio (32); Chr (33) esclamativo; Chr (44) virgola; Chr (46) punto; Chr (63) interrogativo;
If Mid (str, n, 1) = Chr (33) Or Mid (str, n, 1) = Chr (44) Or Mid (str, n, 1) = Chr (46) Or Mid (str, n, 1 )
= Chr (63) Then
'Controlla se lo spazio non è presente
If Mid (str, n + 1, 1) <> Chr (32) Then
` aggiungere lo spazio - notare che viene usato lunghezza della stringa corrente
str = Mid (str, 1, n) & Chr (32) & Right (str, iTxtLen - n)
'Update lunghezza della stringa - incrementa di 1 dopo l'aggiunta di uno spazio
iTxtLen = iTxtLen + 1
End If
End If
Next n
'Cancella gli spazi (se presenti) prima di ogni esclamativo etc e resetta la lunghezza della stringa
iTxtLen = Len (str)
For n = iTxtLen To 1 Step -1
'Chr spazio (32); Chr (33) esclamativo; Chr (44) virgola; Chr (46) punto; Chr (63) interrogativo
If Mid (str, n, 1) = Chr (33) Or Mid (str, n, 1) = Chr (44) Or Mid (str, n, 1) = Chr (46) Or Mid (str, n, 1 )
= Chr (63) Then
'Controlla se lo spazio è presente
If Mid (str, n - 1, 1) = Chr (32) Then
'eliminare uno spazio
str = Application.Replace (str, n - 1, 1, "" )
`ricontrollare nuovamente lo stesso carattere - la posizione di n spostamenti (diminuisce di 1) dovuto alla
cancellazione di un carattere
n = n - 1
End If
End If
Next n
```

Continua -->

```
'maiuscola la prima lettera della stringa e la prima lettera di una parola dopo ogni esclamazione, punto e punto interrogativo, mentre tutte le altre lettere sono minuscole
iStrLen = Len (str)
For i = 1 To iStrLen
'Determinare il codice ANSI di ogni carattere della stringa
ansiCode = Asc (Mid (str, i, 1))
Select Case ansiCode
'97 A 122 sono i codici ANSI equiparano a lettere piccole cap "a" alla "z"
Case 97 To 122
If i> 2 Then
'maiuscola la lettera la cui posizione è 2 caratteri dopo (1 carattere dopo, sarà il carattere di spazio aggiunto in precedenza) un punto esclamativo, punto e il punto interrogativo
If Mid (str, i - 2, 1) = Chr (33) Or Mid (str, i - 2, 1) = Chr (46) Or Mid (str, i - 2, 1) = Chr (63) Then
Mid (str, i, 1) = UCase (Mid (str, i, 1))
End If
'maiuscola la prima lettera della stringa
ElseIf i = 1 Then
Mid (str, i, 1) = UCase (Mid (str, i, 1))
End If
'Se è maiuscola, passare al carattere successivo (es. next i)
Case Else
GoTo salta
End Select
salta:
Next i
'Stringa manipolata
StringManipulation = str
End Function

Sub Str_Man ()
'specificare la stringa di testo per manipolare e ottenere la stringa manipolato
Dim strText As String
'Specificare la stringa di testo, che deve essere manipolato
strText = ActiveSheet.Range ( "A1" ). Valore
'La stringa di testo manipolata viene inserita in A5 del foglio attivo
ActiveSheet.Range ("A5"). Value = StringManipulation (strText)
End Sub
```

L'istruzione GoTo

Si utilizzare l'istruzione GoTo per passare a una linea all'interno della procedura e questa istruzione si compone di 2 parti:

- La dichiarazione GoTo che è la parola chiave GoTo seguita da una etichetta che è l'identificatore
- L'etichetta che è costituita dal nome della stessa seguita da due punti, e poi ha una riga di codice.

Se viene soddisfatta una condizione, con l'istruzione goto viene trasferito il controllo ad una linea separata del codice all'interno della procedura, identificato dall'etichetta. L' istruzione Goto è di solito evitabile se c'è una soluzione alternativa, molte volte è possibile utilizzare If ... Then ... Else o Select Case, in quanto GoTo rende il codice poco leggibile e un po' confuso. Si consiglia di utilizzarlo per la gestione degli errori, vale a dire. "On Error GoTo".

Programmazione ad Oggetti: Metodi e Proprietà

E' bene ricordare che il VBA è un linguaggio di programmazione *orientato agli oggetti* che consente di gestire le applicazioni con semplicità e *la Programmazione ad Oggetti* è un sistema di organizzazione del codice di un programma mediante raggruppamento all'interno di oggetti, ovvero singoli elementi che presentano *Proprietà* e *Metodi*, in pratica non è più il programma a gestire il flusso di istruzioni, ma sono gli oggetti che interagiscono tra di loro intervenendo sulle azioni del programma, introducendo il concetto di Evento che tratteremo nella prossima lezione. Per fare un esempio concreto, possiamo dire che tutto il mondo intorno a noi è pieno di oggetti: la tastiera, il mouse, lo schermo e così via, quello che contraddistingue un oggetto è una serie di caratteristiche o Proprietà, infatti il mouse è dotato di una rotellina e di due tasti, un oggetto, oltre alle proprietà, possiede anche dei Metodi ossia delle funzioni attraverso le quali esercitare delle azioni.

Nell'esempio appena citato, (quello del mouse) tenendo premuto il tasto sinistro e facendo scorrere il mouse si seleziona una o più parole di una riga di testo. Riferendoci alla nostra cartella di lavoro possiamo dire che essa è costituita da oggetti di varia natura, come gli oggetti grafici o i controlli (finestre di dialogo, pulsanti ecc.) incollati su un foglio, ma non è questa la peculiarità di un foglio elettronico la cui ossatura è costituita, partendo dal livello più basso, da :

- celle
- intervalli
- fogli
- cartelle di lavoro

Se osserviamo questo insieme di oggetti come un albero genealogico il capostipite, o il culmine, spetta all'oggetto, denominato *Application*, che rappresenta, tutti gli Oggetti di Excel. Si tratta appunto di Excel stesso. Proseguendo nella discesa genealogica troveremo i seguenti oggetti:

- L'oggetto Workbook : Che è la cartella di lavoro (cioè il nostro file)
- L'oggetto Worksheet : Che è il foglio di lavoro (Foglio1, Foglio2 ecc...)
- L'oggetto Range : Che è un intervallo di celle (A1: B12, C1:D12, ecc...)

Sintatticamente possiamo affermare che ciascun oggetto fa parte di una famiglia o classe e l'accesso al singolo membro di ciascuna classe si effettua attraverso metodi, *pluralistici*, cioè da una pluralità di metodi che collaborano e interagiscono con i vari oggetti in maniera omogenea e ai quali corrispondono insiemi di oggetti quali :*Workbooks*, *Worksheet*, *Range* e *Cells*, inoltre i membri più elevati si possono omettere nel caso che il soggetto sia attivo; vedremo meglio questo passaggio fra poche righe.

Fin qui abbiamo delineato i componenti principali cercando di esporre come vengono interpretati dal Visual Basic applicato al foglio elettronico, ma l'obiettivo vero è quello di focalizzare gli oggetti che formano l'ossatura, il nucleo di uno *spreadsheet* (foglio di lavoro), al cui centro, vi sono intervalli e celle con il loro contenuto di dati da elaborare o formule e il loro inquadramento nel mondo Visual Basic è fondamentale e aiuterà a capire meglio tutto il resto, vediamo di interpretare quanto appena affermato usando il Visual Basic. Con queste sintassi

Workbooks("Lezione2.xls") e *Worksheets("Foglio1")*

Si individuano rispettivamente la cartella e il foglio di lavoro (notare i loro nomi virgolettati dentro le parentesi), ma dobbiamo però tenere presente che un elemento può anche venire individuato tramite un indice, il quale può essere o il numero di ordine o il nome fra virgolette, per capire meglio il concetto di

indice possiamo dire che la sintassi Workbooks(2) e Workbooks("Lezione2.xls") puntano entrambe alla stessa cartella Lezione2.xls a patto che questa sia la **seconda** fra quelle aperte contemporaneamente da Excel.

Infatti se abbiamo solo la cartella Lezione2.xls aperta, la sintassi esatta diventa *Workbooks(1)*. A questo punto è abbastanza chiaro che l'indice che usiamo fra parentesi nell'oggetto Workbooks varia ed è strettamente legato al numero di cartelle aperte nel momento dell'esecuzione di questa istruzione, pertanto possiamo far notare che è possibile scrivere questa istruzione in tre diversi modi:

- *Application.Workbooks(1).Worksheets(1).Range("A1:B 10")*
- *Application.Workbooks("Lezione2.xls").Worksheets(1).Range("A1:B10")*
- *Application.Workbooks("Lezione2.xls").Worksheets(" Foglio1").Range("A1:B10")*

Ricordate che poco sopra abbiamo però affermato che i membri più elevati si possono omettere quando sono attivi, per cui:

- Se abbiamo Excel aperto
- Se la cartella Lezione2.xls è aperta
- Se infine ci troviamo nel Foglio1

Possiamo ridurre tutto il listato ad un semplice Range("A1:B10"), in caso contrario credo che sia abbastanza chiaro come agire usando gli indici per individuare il nostro intervallo (o Range). Si deve inoltre prestare attenzione alle proprietà "pluralistiche" che poco sopra abbiamo citato: le prime volte è facile scordarsi del plurale, scrivendo *Workbook*("Lezione2.xls") anziché Workbooks("Lezione2.xls") oppure *Worksheet*("Foglio1") invece diWorksheets("Foglio1").

Vediamo qualche esempio di codice da applicare a quanto esposto finora, ma sopratutto vediamo come automatizzare l'esecuzione di routine creando un pulsante, ad esempio apriamo la nostra cartella ("Lezione2.xls") e rientriamo nel Visual Basic Editor. Questa volta scriveremo noi una macro direttamente senza affidarci all'aiuto del Registratore di macro utilizzato nella precedente lezione. Ponendoci al di sotto dell'ultima riga della macro precedente (basta porre il cursore alla fine di End Sub e premere una o due volte Invio), scriviamo il seguente codice

```
Sub Nascondi_Foglio()
Worksheets(2).Visible = False
MsgBox "Il Foglio 2 è sparito"
End Sub
```

Commentiamo questo codice: La routine inizia con *Sub* (seguito dal nome della routine) e termina con *End Sub*. L'oggetto Worksheets(2) ossia il Foglio 2, è seguito da una sua proprietà che si evidenzia con *.visible*, si richiama perciò la proprietà *visible* che consente al foglio di essere visibile o meno a seconda del fatto che tale proprietà sia dichiarata *vera* [True] o *falsa* [False] (nel nostro caso è stata dichiarata uguale a False per cui indica che il foglio sarà nascosto).

MsgBox "Il Foglio 2 è sparito" è invece una funzione di VBA che permette di mostrare all'utente un messaggio che nella sua forma più semplice riporta un messaggio di avviso per l'utente (il testo racchiuso fra virgolette) ed un bottone di OK necessario per la sua chiusura. Torniamo ora ad Excel e facciamo click col destro del mouse sulla barra degli strumenti, ci comparirà un box con delle voci, scegliamo *Moduli*

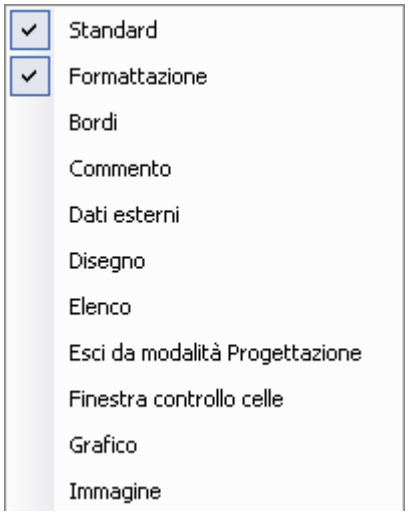


Fig. 1

Ci comparirà un nuova barra flottante che possiamo ancorare alle altre barre degli strumenti semplicemente trascinandocela.

Fig. 2

Premiamo sul pulsante evidenziato dalla freccia rossa e poi spostandoci sul foglio di lavoro teniamo premuto il pulsante di sinistra del mouse, trasciniamolo fino a raggiungere le dimensioni desiderate ed al rilascio ci comparirà il nostro pulsante. Contestualmente si aprirà anche la finestra per assegnare la macro (Nascondi_Foglio) al pulsante appena creato

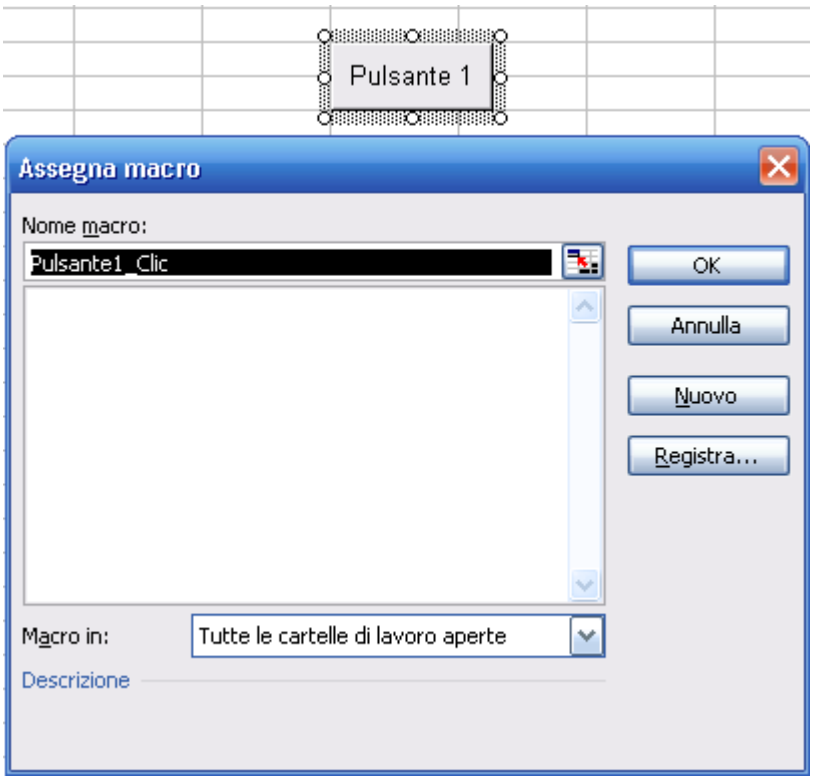


Fig. 3

Stessa operazione e identico risultato si può ottenere inserendo un'immagine sul foglio (si dovrà attivare la barra degli strumenti *Disegno*) e quindi tramite il solito menu contestuale ottenuto cliccando col destro sull'immagine, assegnargli la macro. Possiamo ridimensionare il pulsante, editarne il testo, cambiarne i caratteri etc. facendo clic sopra al pulsante stesso col pulsante destro del mouse e scegliere, tra le voci del menù contestuale che compare, quella di cui abbiamo bisogno. Se invece vogliamo spostare il pulsante, facciamo ancora clic col destro del mouse, sparirà il menù contestuale e rimarrà evidenziato il pulsante

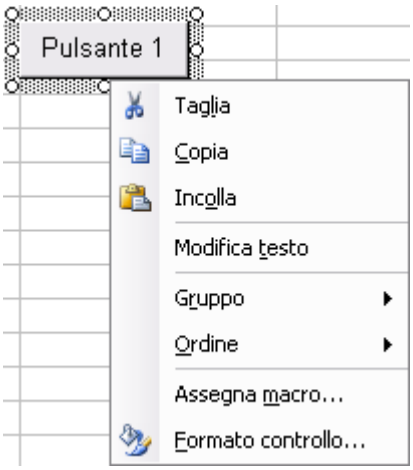


Fig. 4

Possiamo allora spostarlo a piacimento e ridimensionarlo trascinandolo negli angoli

Fig. 5

NB **Per versioni di Excel superiori alla 2003** *si deve seguire il percorso:* ***File - Opzioni - Personalizzazione barra multifunzione*** *e nel box Personalizza barra Multifunzione mettere il flag al campo Sviluppo, come da figura sottostante*

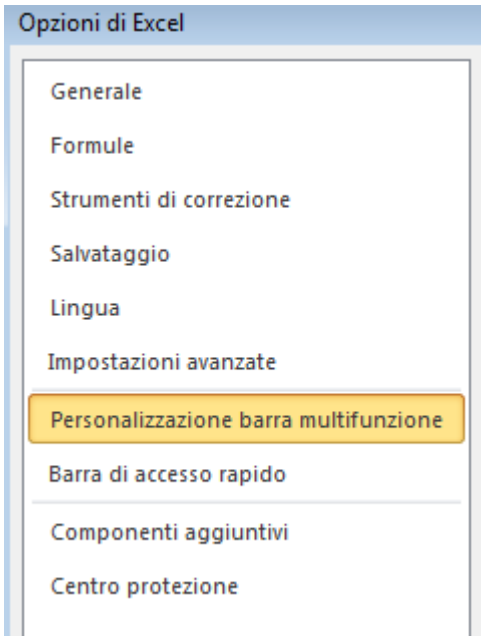


Fig. 6

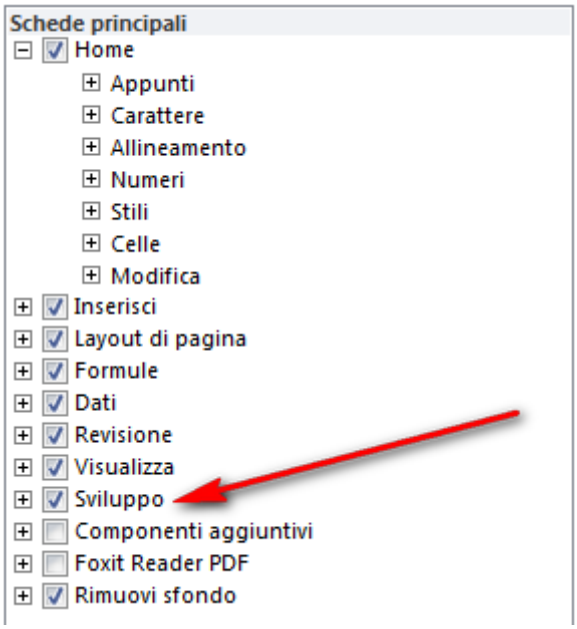


Fig. 7

A questo punto è comparso un altro menu nella barra multifunzione, il menu *Sviluppo*, tramite il sotto menu *Inserisci* scegliere il pulsante e procedere come spiegato poco sopra per la Ver. 2003

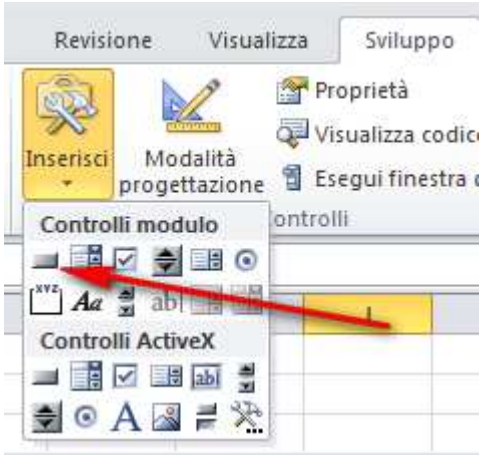


Fig. 8

PS: E' possibile usare una forma geometrica invece di un pulsante a cui assegnare una macro, in questo caso si procede in questo modo: dal menu **Inserisci - Forme**

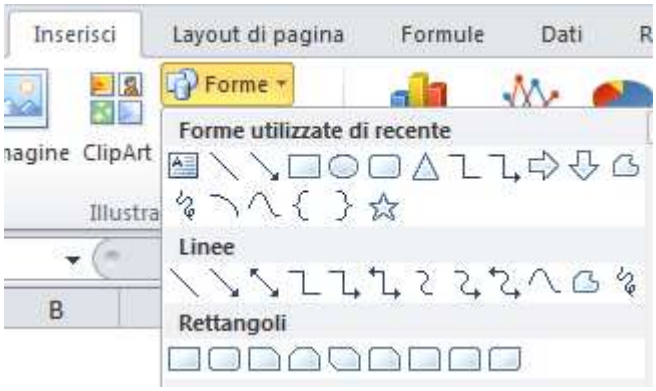


Fig. 9

Scegliere il rettangolo e disegnarlo sul foglio, a operazione compiuta verrà riportata in automatico la finestra delle macro, in alternativa è possibile farla comparire cliccando col tasto destro del mouse sulla forma e nel menù che compare scegliere la voce *Assegna Macro*

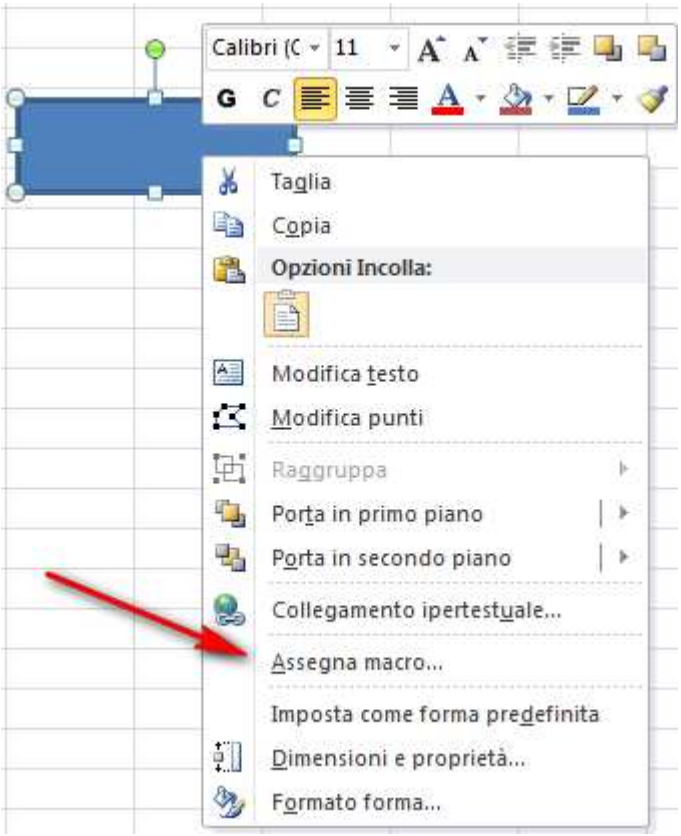


Fig. 10

Fine Nota

Possiamo ora mandare in esecuzione la macro stessa semplicemente cliccando sul pulsante appena creato, adesso il foglio è sparito, come facciamo a farlo riapparire? Basta creare una nuova macro che assoceremo ad un altro pulsante, con le stesse modalità di costruzione della precedente:

```
Sub Mostra_Foglio()
Worksheets(2).Visible = True
MsgBox "Il Foglio 2 è Ricomparso"
End Sub
```

I commenti sono superflui, il codice è esattamente identico al precedente solo che la proprietà **visible** è stata ora posta a **True** (Vero) ed in tal caso il Foglio 2 riappare.

Ambiente di sviluppo e generatore di macro

Avrete certamente notato, usando quotidianamente Excel, che capita molto spesso di svolgere le stesse operazioni ripetendo lo stesso comando per compiere normalissime azioni di routine che implicano le stesse modalità di esecuzione. Invece di ripetere la stessa serie di istruzioni ogni volta che si vuole eseguire una determinata operazione è possibile, utilizzando una macro, creare un automatismo che ci permetta di eseguire una serie di comandi automaticamente all'interno dell'applicazione stessa.

Le macro sono procedure, o meglio, un insieme di comandi e istruzioni, che permettono di eseguire una serie di operazioni utilizzando un singolo comando, automatizzando attività ripetitive che altrimenti richiederebbero una vasta serie di comandi manuali in sequenza. In pratica una macro è una procedure composta da un insieme di istruzioni che permettono di eseguire un compito ripetitivo e che Microsoft ha reso disponibile nei suoi pacchetti Office anche ai non addetti ai lavori attraverso l'introduzione del *Registratore di Macro*

Il Registratore di macro

Un modo semplice per imparare VBA è quello di registrare le proprie macro con lo strumento Registratore di macro, e quindi leggere, eseguire e modificare il codice della macro che avete registrato in Visual Basic Editor. Una macro è un insieme di istruzioni fornite sotto forma di codici per rendere il computer in grado di eseguire un'azione e questo particolare strumento è presente in tutte le applicazioni di Office e permette di registrare una serie di operazioni in una macro per poi essere eseguite quando se ne presenta la necessità.

Bisogna tenere presente che una volta attivato il registratore di macro vengono registrate *tutte le azioni svolte dall'utente* e convertite in codice Visual Basic, pertanto prima di compiere questa azione si devono fissare le condizioni nelle quali successivamente queste macro verranno eseguite, in sostanza, si deve stabilire nel proprio ambiente di lavoro le stesse condizioni che esisteranno quando la macro verrà eseguita. Questa operazione viene definita col termine di *fissare le condizioni iniziali*, per meglio comprendere si può supporre, per esempio, di voler creare una macro che applica un determinato tipo e dimensione di carattere a delle celle selezionate di un foglio Excel, le condizioni iniziali, che abbiamo accennato prima, sono che questa macro abbia almeno un foglio di calcolo aperto e una cella selezionata.

Se avviamo il registratore di macro, dopo visualizziamo un foglio di calcolo e selezioniamo delle celle, tutte queste operazioni entrano a far parte della macro stessa in quanto, come abbiamo già detto, il registratore di macro registra tutte le azioni che si eseguono, anche quelle che esulano dal compito che si vuole assegnare, diventando così parte integrante della macro. In pratica per definire le condizioni iniziali si intende la "*preparazione dell'ambiente di lavoro*" in modo che il codice della macro contenga esclusivamente le operazioni da svolgere. Per poter mandare in esecuzione una macro dobbiamo prima verificare il grado di protezione che abbiamo impostato in Excel, lo possiamo vedere attraverso questo percorso **Strumenti - Macro - Protezione** e ci comparirà una maschera come quella sotto riportata

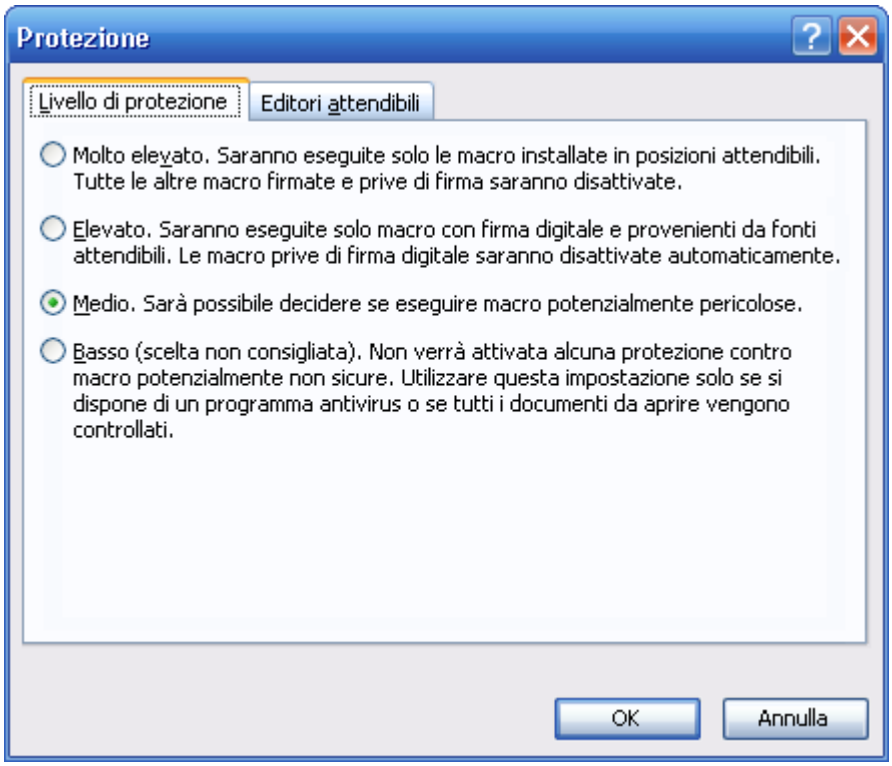


Fig. 1

Impostando il livello di protezione su *medio*, come vedete in *Figura 1*, verrà mostrato un messaggio di avviso ogni volta che andrete ad aprire file che conterranno macro, e permetterà di scegliere se eseguirle o meno. Il messaggio che vi apparirà è il seguente

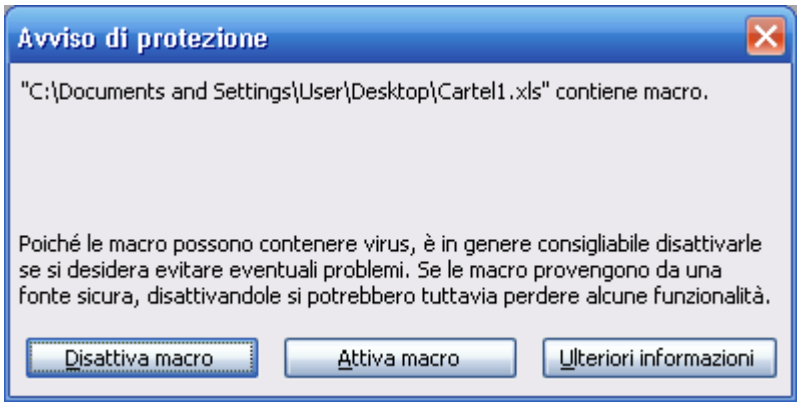


Fig. 2

NB **Per versioni di Excel superiori alla 2003** il livello di sicurezza è settato per default con le macro disabilitate, per poter salvare il file con il progetto VBA si deve seguire questo percorso dal menu **File - Salva con nome** e selezionare nella casella a discesa la voce *Cartella di lavoro con attivazione macro di Excel*

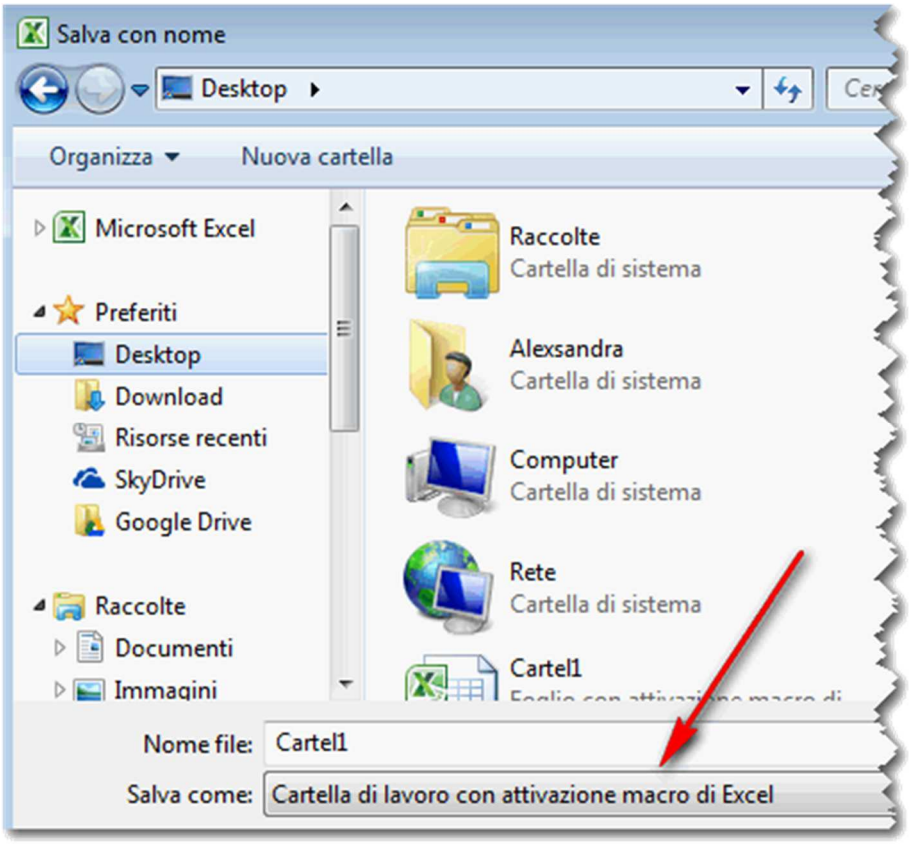


Fig. 3

Ad ogni modo si consiglia di controllare il **Centro di protezione** per verificare il grado di sicurezza impostato a cui si può accedere dalla barra multifunzione seguendo il percorso dal menu **Sviluppo - Sicurezza - Macro**

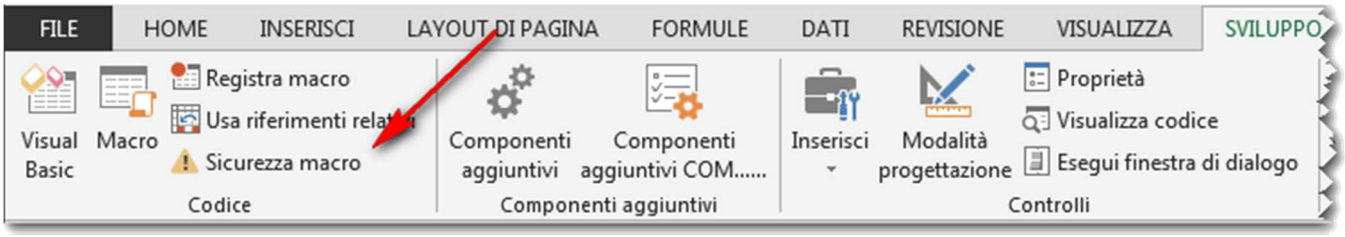


Fig. 4

Successivamente viene visualizzata la finestra del centro di protezione come quella sotto riportata

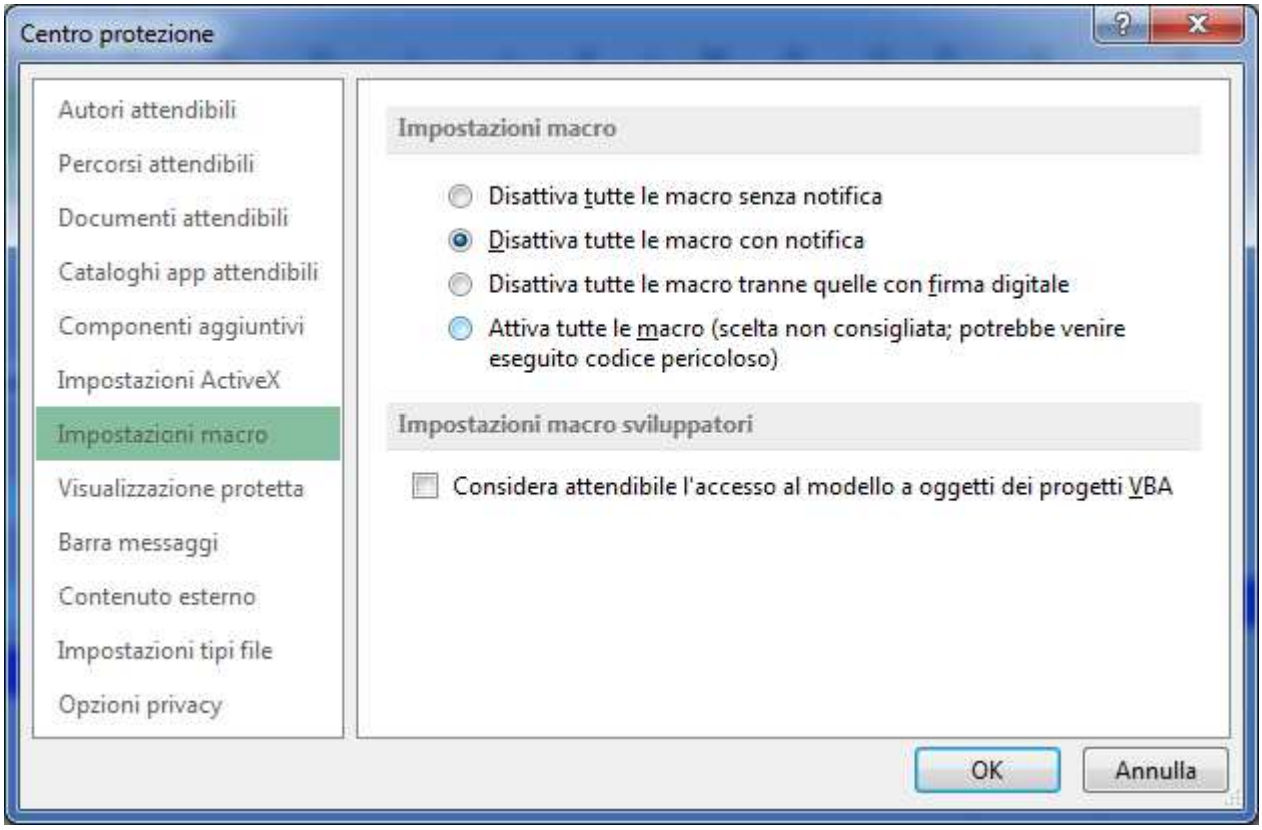


Fig. 5

Si consiglia di selezionare l'opzione "*Disattiva tutte le macro con notifica*" in questo modo quando viene aperto un file contenente codice VBA Excel rimanda un avviso che il file che si sta per aprire contiene un progetto VBA che identifica come potenzialmente pericoloso. Scegliendo questa opzione quando poi andiamo ad aprire un file che contiene macro ci viene portato a video sotto la barra dei menù un avviso in cui possiamo scegliere se attivarle oppure no

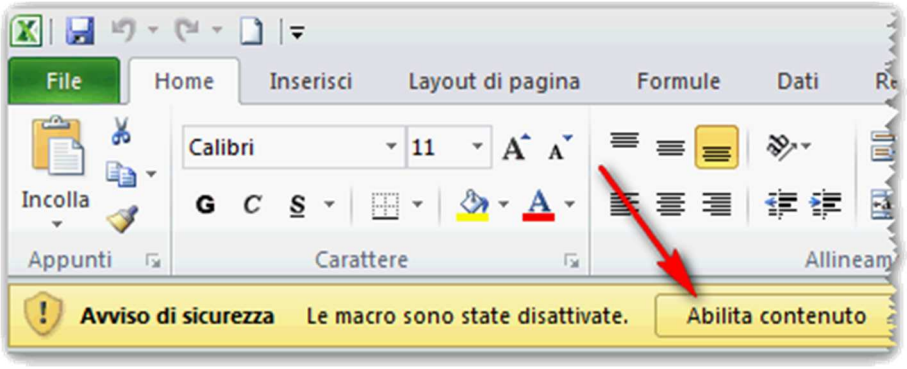


Fig. 6

E' doveroso ricordare che le macro **possono contenere codice dannoso** o anche un'applicazione virale, pertanto fate attenzione nell'aprire file che riporteranno il messaggio sopra esposto, aprite file contenenti macro solo se siete certi della fonte.

Fine Nota

Fatta questa premessa iniziamo a costruire la nostra prima macro utilizzando il registratore di Macro che si trova nel menu **Strumenti - Macro - Registra nuova macro**

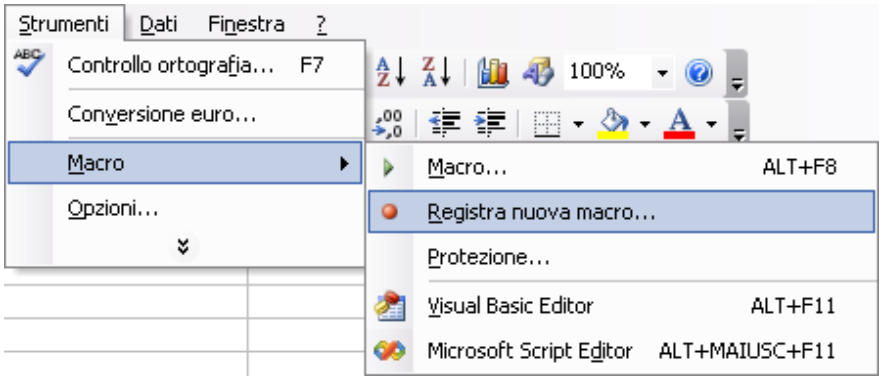


Fig. 7

Per versioni di Excel superiori alla 2003 il percorso che dobbiamo seguire per registrare una nuova macro è: **Visualizza - Macro - Registra Macro**

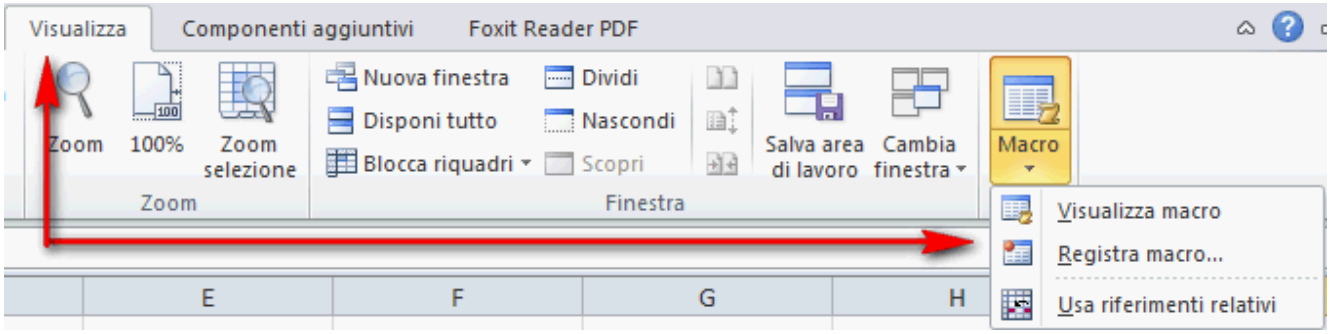


Fig. 8

Fine Nota

Eseguita questa operazione ci comparirà una finestra come quella sotto riportata

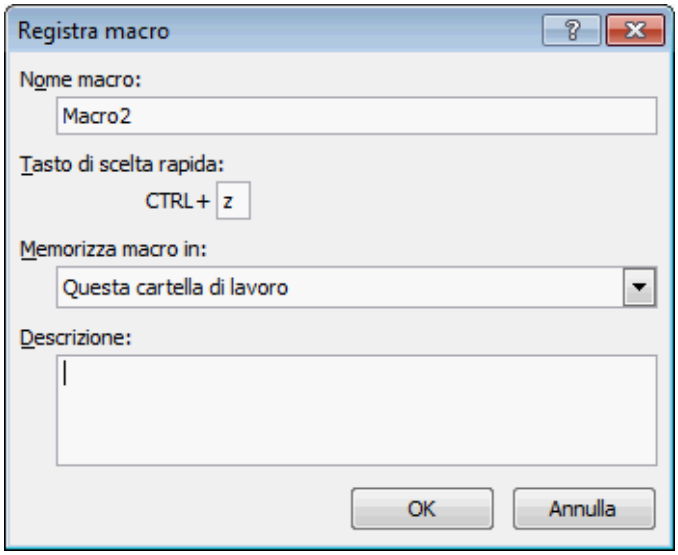


Fig. 9

Vediamo i vari campi che compaiono in questa finestra

Nome macro: La prima opzione della finestra di dialogo è il Nome Macro che per default viene assegnato un nome che consiste nella parola Macro seguita da un numero in funzione del numero di macro già registrate in quella sessione di lavoro. Si consiglia però di assegnare un nome più significativo e personalizzato per identificare il compito assegnato alla macro. Per esempio se si registra una macro che cambia il tipo e la dimensione del carattere si può immettere il nome *Cambiacaratt* nel campo di testo del nome della macro

Tasto di scelta rapida: si può usare facoltativamente la casella di immissione rapida da tastiera per definire una combinazione di tasti che quando verrà premuta porterà all'esecuzione della macro, immettendo quindi la lettera che si vuole usare come "*tasto rapido*". Si consiglia di usare questa opzione solo nel caso in cui si ritenga che in seguito la macro verrà usata molto spesso. Ricordo inoltre che tutte le combinazioni di tasti rapidi o "*scorciatoie*" sono combinazioni tra il tasto Ctrl (Control) e il tasto che viene assegnato, per esempio, se inseriamo Z nell'apposito campo, per eseguire la macro basta usare la combinazione di tasti *CTRL+Z*

Memorizza macro in: In questo campo di testo è possibile scegliere dove salvare la macro una volta registrata. Le scelte previste sono:

- Cartella macro personale
- Nuova cartella di lavoro
- Questa cartella di lavoro

Se si sceglie *Cartella macro personale* Excel salva la macro in una speciale cartella detta *Personal.xls* che viene caricata automaticamente quando si avvia Excel. Se invece si sceglie "*Questa cartella di lavoro*", la macro verrà salvata nella cartella corrente (scelta consigliata)

Descrizione: Le informazioni inserite in questo campo non vengono usate dalla macro ma servono solo per aggiungere alcune note o commenti su ciò che fa la macro, per default il VBA immette una descrizione costituita dalla data di registrazione della macro e dall'autore

Per fornire un esempio specifico, si supponga che tutti i giorni si debba aprire un file che si chiama "*Lezione1*" e si deve scrivere nella cella A1 del Foglio1*Inizio ore 08,00* e nella cella A1 del Foglio2 *Fine ore 12,00* potremmo automatizzare questa procedura usando una macro. Premiamo sul tasto OK della figura sopra esposta (**Fig. 9**) e procediamo alla creazione della macro. Innanzi tutto attiviamo il *Foglio1* e portiamo il cursore nella cella A1 e scriviamo la frase *Inizio ore 08,00* e poi premiamo il tasto Invio, portiamoci poi nel *Foglio2* e nella cella A1 e scriviamo la frase *Fine ore 12,00* e premiamo ancora Invio

Ritorniamo poi al *Foglio1* e possiamo anche interrompere la registrazione della nostra macro seguendo questo percorso: **Strumenti - Macro - Interrompi registrazione**

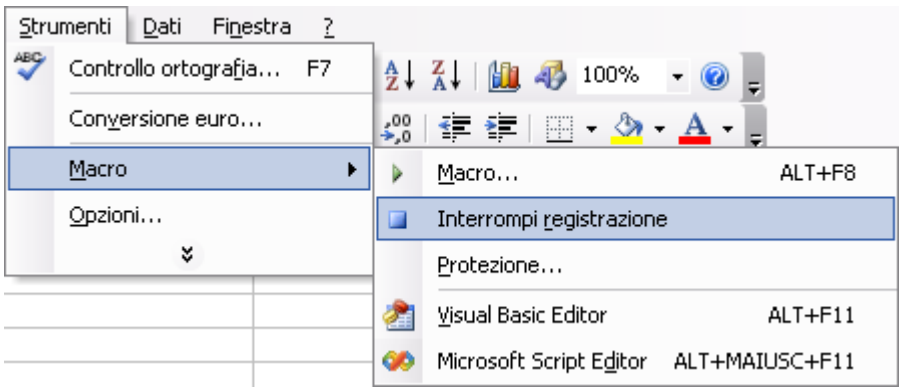


Fig. 10

A questo punto la nostra macro è stata creata, la procedura di fine registrazione può essere eseguita nel modo sopra descritto oppure al momento della registrazione può comparire a video anche un box come il seguente

Fig. 11

La finestra di *figura 11* rimarrà sullo schermo durante la registrazione, è possibile spostarlo come volete, e per fermare la registrazione basta premere sul pulsante quadrato che vedete in figura.

NB **Per versioni di Excel superiori alla 2003** per interrompere la registrazione della macro si deve seguire il seguente percorso **Visualizza - Macro - Interrompi Registrazione** e la finestra riportata in *figura 11* si chiude

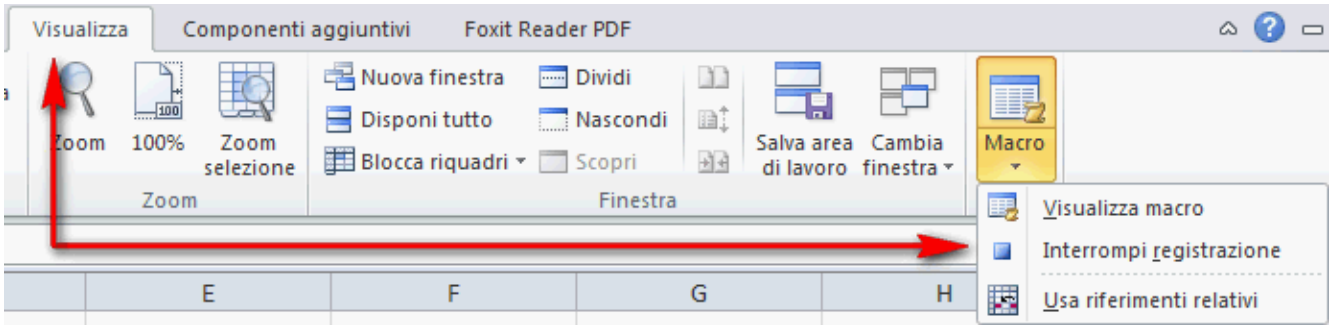


Fig. 12

Fine Nota

A questo punto possiamo verificare se la macro registrata funziona, cancelliamo quello che abbiamo scritto nei due fogli e proviamo a mandarla in esecuzione, seguendo il percorso dal menu **Strumenti - Macro - Macro**

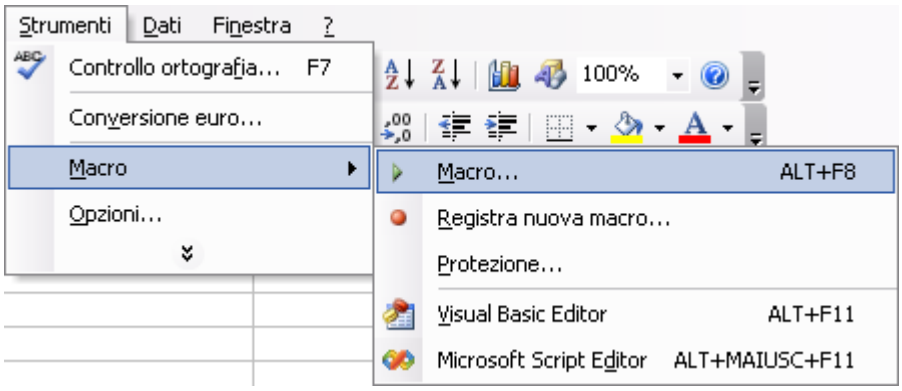


Fig. 13

NB **Per versioni di Excel superiori alla 2003** per mandare una macro in esecuzione il percorso da seguire è: **Visualizza - Macro - Visualizza Macro**

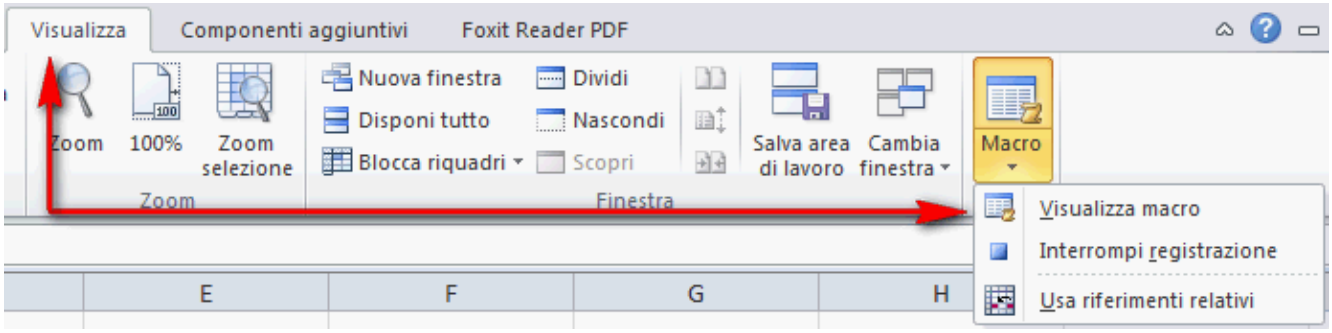


Fig. 14

Fine Nota

In seguito si apre una maschera come la seguente

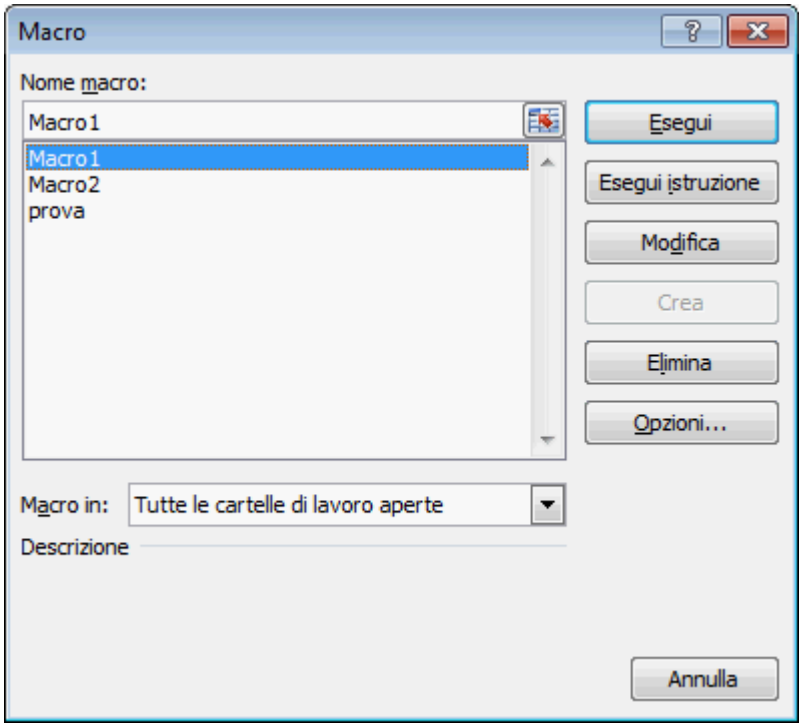


Fig. 15

In questo box vengono elencate tutte le macro registrate o scritte direttamente all'interno dell'editor, basta solo selezionare col mouse la macro interessata e premere sul pulsante *Esegui* per mandarla in esecuzione, oppure utilizzando il tasto di scelta rapida associato alla macro (CTRL+Z) o semplicemente premendo il tasto *F5*. Si può eseguire una macro in modalità interruzione, (o Step by Step) selezionando la macro e cliccando sul pulsante "*Esegui Istruzione*" che aprirà la finestra dell'Editor con il modulo che contiene la macro selezionato e già attivo e col cursore posizionato sulla prima riga di codice selezionata in giallo. Le istruzioni verranno eseguite premendo il tasto *F8*, questa è un'opzione per poter verificare il funzionamento della macro passo-passo. Premendo il tasto "*Elimina*" la macro selezionata verrà eliminata e cliccando sul tasto "*Opzioni*" si può cambiare l'assegnazione del tasto di scelta rapido e la descrizione della macro.

Cliccando invece su "*Modifica*" si visualizza il codice della macro registrata, e si accede all'Editor di VBE con la finestra dell'Editor già aperta e il codice della macro che è stato prodotto dall'azione compiuta. Nel nostro esempio troveremo un codice come il seguente:

```
Sub Macro1()
'
' Macro1 Macro
' prova registratore di macro
'
'
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "Inizio ore 8,00"
    Range("A2").Select
    Sheets("Foglio2").Select
    ActiveCell.FormulaR1C1 = "Fine ore 12,00"
    Range("A2").Select
    Sheets("Foglio1").Select
End Sub
```

Analizzando il codice vediamo che la prima riga contiene l'intestazione della procedura, cioè la parola chiave *Sub* seguita dal nome che abbiamo assegnato alla macro e da due parentesi, che indica l'inizio della sub routine e delle istruzioni che verranno eseguite quando manderemo in esecuzione la macro. Nell'ultima riga è presente la parola chiave *End Sub* che indica la fine della procedura e tra le due istruzioni vengono collocate le istruzioni da seguire

Le prime 5 righe sono precedute da un apice e sono dei commenti che vengono ignorate dal VBE, praticamente sono le frasi che abbiamo inserito in fig. 15 all'inizio della registrazione, mentre alla riga 6 viene selezionata la cella A1 tramite l'istruzione 'Select' e nella riga successiva viene inserita la scritta "Inizio ore 8,00" mentre alla riga successiva viene selezionata la cella A2 che corrisponde alla pressione del tasto Invio che rimanda a capo e seleziona la cella A2 nel foglio di lavoro.

Nella riga 9 viene selezionato il foglio 2 e nella riga successiva viene inserita la scritta "Fine ore 12,00" nella cella A1, si passa poi alla cella A2 quando viene premuto il tasto invio nel foglio di lavoro che rimanda a capo e infine si ritorna al foglio 1 terminando la registrazione.

Se si desidera che una macro selezioni una cella specifica, esegua un'azione e poi selezioni un'altra cella mediante un riferimento relativo alla cella attiva, è possibile utilizzare dei riferimenti di cella definiti *relativi* e *assoluti* durante la registrazione della macro. Per registrare una macro utilizzando riferimenti relativi, si deve attivare la funzione dal percorso **Visualizza – Macro – Usa riferimenti relativi** della barra multifunzione

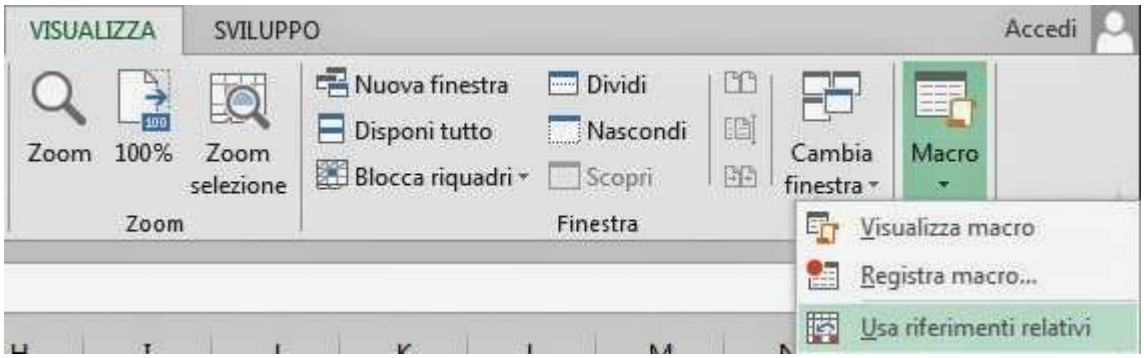


Fig. 16

Definizione di "riferimento assoluto"

Per "*riferimento assoluto*" si intende l'indirizzo esatto di una cella, indipendentemente dalla posizione della cella contenente la formula. Un riferimento assoluto ha la forma $A$1, in pratica viene inserito il simbolo $ prima del riferimento, notare che possono esistere riferimenti di cella misti, che presentano la seguente forma A$1, in questo caso il riferimento assoluto è riferito alla riga, mentre un riferimento assoluto per la colonna ha la forma $A1. Diversamente dai riferimenti relativi, i riferimenti assoluti non si adattano automaticamente quando si copiano delle formule su righe e colonne.

```
Sub copia()
' copia nella cella A1 il contenuto di A3 con indirizzamenti assoluti
 Range("A1").Value = Range("A3").Value
End Sub
```

Definizione di "riferimento relativo"

Per "*riferimento relativo*" si intende un riferimento di cella, utilizzato in una formula, che cambia quando la formula viene copiata in un'altra cella o intervallo. Dopo che la formula viene copiata e incollata, il riferimento relativo nella nuova formula cambia per fare riferimento a una cella differente distante dalla formula lo stesso numero di righe e colonne che distanziano il riferimento relativo originale dalla formula originale.

Se ad esempio la cella A3 contiene la formula =A1+A2 e si copia la cella A3 nella cella B3, la formula nella cella B3 diventa *=B1+B2*. Mentre invece la formula *"=R[-1]C*2*" è un esempio di riferimento relativo e si riferisce alla riga precedente [-1] a quella attiva e la stessa colonna.

Limitazioni del Registratore di macro

Si può osservare che la procedura di registrazione di una macro, anche se è un modo semplice e uno strumento utile per imparare e creare codici VBA, può essere utilizzato solo per i *codici semplici* e di base e non per la creazione di codici avanzati e procedure complesse a causa di alcune limitazioni. Durante la registrazione di una macro è possibile creare solo sotto-procedure e non le funzioni che restituiscono un valore, non è possibile alloccare in memoria le informazioni durante la registrazione e non possono essere utilizzate le variabili, oppure utilizzare istruzioni condizionali quali If-Then, o utilizzare cicli Loop, o chiamare altre procedure o funzioni. Un codice di una macro registrata è di solito inflessibile e non è il più efficace, anzi necessita sempre di una pulizia nelle linee di codice dalle informazioni inutili aggregate alla macro registrata.

Vantaggi del registratore di macro

1. Permette di registrare molto velocemente una sequenza di istruzioni, che scrivendole direttamente richiederebbe più tempo.
2. E' molto utile quando non si ricorda una determinata istruzione, oppure una sequenza particolare per compiere una determinata operazione, è sufficiente registrare una macro che esegue quell'azione e poi andare a vedere come è stata convertita in codice sorgente per raccogliere l'istruzione chiave e utilizzarla nelle proprie macro articolate!

Azioni ripetitive: Il Ciclo Do Loop

Abbiamo parlato di due tipi di cicli, quelli ad *interazione fissa* e quelli ad *interazione indefinita*, il ciclo **Do ..Loop** appartiene ai cicli ad interazione indefinita, VBA ci fornisce questa istruzione estremamente potente per costruire strutture cicliche indefinite nelle nostre funzioni o procedure. Essenzialmente è costituito da una singola istruzione: Do. Questa istruzione ha molte opzioni ed è talmente flessibile che ci fornisce quattro diverse possibilità per costruire dei cicli raggruppati in due diverse categorie di base, che sono i cicli controllati da un contatore e i cicli controllati da eventi, quale è la differenza?

In un ciclo controllato da un contatore, le istruzioni del corpo del ciclo vengono eseguite finchè il valore è inferiore o superiore al limite specificato, in sostanza non varia molto dal ciclo For ..Next, eccetto che il programmatore è direttamente responsabile per l'inizializzazione della variabile contatore e per l'incrementazione o decrementazione del contatore. Potremmo usare il ciclo Do se il passo del contatore non è regolare, o se non c'è modo di determinare il limite finale se non dopo che il ciclo ha iniziato la sua esecuzione. Per esempio se vogliamo spostarci attraverso 15 righe di un foglio, alcune volte avanzando di una sola riga e altre volte avanzando di due righe, poichè il numero di righe da avanzare (cioè il passo del contatore) cambia, non possiamo usare il ciclo *For ..Next* ma dobbiamo usare il ciclo Do

Per i cicli controllati da eventi, le istruzioni vengono eseguite quando la determinante del ciclo diventa vera o falsa sulla base di alcuni eventi che si verificano all'interno del corpo del ciclo. Per esempio potremmo scrivere un ciclo che viene eseguito indefinitamente fino a quando l'utente non inserisce un particolare valore in una finestra di dialogo input, e l'inserimento di questo particolare valore è l'evento che termina il ciclo, oppure possiamo eseguire delle operazioni sulle celle di un foglio fino a quando non si raggiunge la cella vuota di una colonna, anche in questo caso il raggiungimento della cella vuota è l'evento che termina il ciclo. Per ora abbiamo chiarito le due categorie di cicli, abbiamo capito che esistono cicli controllati da un contatore e cicli controllati da eventi, vediamo ora la sintassi

Do
istruzioni
Loop Until *condizione*

e qualche esempio.

```
Sub ciclo1()
Dim a As Integer
Do
 a = a + 1
 MsgBox (a)
Loop Until a = 10
End Sub
```

Questo è un ciclo controllato da un contatore, noterete che c'è poca differenza dal ciclo For ..Next , infatti otteniamo lo stesso effetto, cioè portiamo a video un messaggio col valore della variabile finchè non arriva a 10, ma abbiamo visto poco sopra che abbiamo quattro diverse possibilità di costruire cicli divisi in due categorie, ora le categorie le abbiamo viste *(cicli controllati da un contatore e i cicli controllati da eventi)* vediamo ora i quattro modi di costruire un ciclo, il listato sopra esposto è un metodo, vediamo ora gli altri e dopo li commentiamo assieme

```
Sub ciclo2()
Dim a As Integer
Do Until a = 10
 a = a + 1
 MsgBox (a)
Loop
End Sub

Sub ciclo3()
Dim a As Integer
Do
a = a + 1
MsgBox (a)
Loop While a <> 10
End Sub

Sub ciclo4()
Dim a As Integer
Do While a <> 10
a = a + 1
MsgBox (a)
Loop
End Sub
```

Come potete vedere la differenza sta nell'enunciato di dichiarazione del ciclo, il risultato non cambia ma le parole chiave e la loro collocazione si, vediamo come viene interpretato l'enunciato : *Do Until a = 10* [Ripeti finchè a = 10], oppure *Do While a <> 10* [Ripeti finchè a è diverso da 10], in questi 2 casi la differenza fondamentale è che viene verificata la condizione determinante *prima* che venga eseguito il ciclo, infatti se a fosse = a 20 il ciclo non verrebbe eseguito, gli altri 2 metodi Do .....Loop While a <> 10 e Do ..... Loop Until a = 10 vengono interpretati come spiegato sopra, ma la determinate del ciclo viene verificata alla fine del ciclo.

In pratica la forma *Do ... Loop While* e *Do ... Loop Until* prima vengono eseguite le istruzioni presenti nel ciclo e poi quando raggiunge la parola chiave *Loop* viene verificata la condizione (vedi sintassi) se condizione è *False* (nel caso si usi Until VBA ritorna all'inizio del ciclo ed esegue nuovamente le istruzioni del ciclo, se invece usiamo la forma While la condizione da verificare deve essere *True*, mentre nelle altre 2 forme espresse Do Until ...Loop e Do While ...Loop la condizione invece viene verificata subito.

Vediamo un esempio che semplifica e chiarifica quanto esposto.
Supponiamo di far comparire a video un Inputbox per chiedere informazioni all'utente, e poi verificare se i dati immessi siano validi, in questo caso inseriamo nel nostro ciclo dobbiamo eseguire le istruzioni (far comparire il messaggio di Input) e all'inserimento dei dati da parte dell'utente verifichiamo la condizione, il listato si presenta così

```
Sub esempio1()
Dim pass As String
Do
pass = InputBox(prompt:="Inserisci password di accesso per Proseguire: ",
Title:="Controllo accessi")
Loop Until pass = "Alex"
MsgBox "Password esatta: Accesso consentito", vbInformation + vbYes, "Verifica Password"
End Sub
```

Se proviamo questo codice vedremmo che il ciclo continua a ripetere le istruzioni finchè non viene digitata la password esatta, però se vogliamo uscire perchè non ricordiamo la password?, anche premendo sul tasto "Annulla" dell'Inputbox le istruzioni continuano ad essere ripetute lo stesso, allora è sempre meglio porre una condizione per evitare di proseguire, ma al tempo stesso controllare che l'esecuzione del ciclo avvenga correttamente, in pratica mettiamo l'utente in grado di uscire da un ciclo senza però che eviti di inserire la password richiesta. Per ovviare a questo modifichiamo il listato in questo modo.

```
Sub esempio2()
Dim pass As String
Do
pass = InputBox(prompt:="Inserisci password di accesso per Proseguire: ",
Title:="Controllo accessi")
If pass <> "Alex" Then Exit Sub
Loop Until pass = "Alex"
MsgBox "Password esatta : Accesso consentito", vbInformation + vbYes, "Verifica
Password"
End Sub
```

Inserendo la riga con il ciclo **IF** abbiamo posto una condizione, cioè *"Se pass è diverso da "Alex" esci dalla sub"*, attenzione, che esci dalla sub non vuol dire proseguire, ma semplicemente abbandoniamo questa routine che in ogni caso per poter proseguire dobbiamo inserire la password giusta, questa semplice metodica viene chiamata *Uscita forzata dal ciclo".*
Anche con l'altro metodo cioè *Do Until ...Loop* e *Do While ...Loop* il risultato non cambia, pertanto non c'è una regola ben precisa su quale forma sia meglio usare, possiamo dire che a disposizione VBA ci mette queste forme, sta a noi usare quella che ci pare più logica o intuitiva per esprimere quello che vogliamo esegua il nostro codice. Se vi ricordate nella lezione precedente avevamo parlato della nidificazione del ciclo IF, anche nei cicli For ... Next e Do ...Loop è possibile eseguirla, visto che l'argomento lo abbiamo già toccato in questo contesto facciamo subito un esempio e dopo lo commenteremo assieme

```
Sub scrivi_col_for()
For riga = 12 To 35
 For colonna = 7 To 18
 Cells(riga, colonna).Value = riga
 Next colonna
Next riga
End Sub
```

Cosa abbiamo fatto? se provate ad eseguire la macro vedrete che nell'area di lavoro abbiamo riempito tutte le celle con il numero della riga corrispondente. Vediamo il codice *For riga = 12 To 35* il contatore del ciclo è rappresentato dalla variabile riga e gli diciamo *[Per riga che và da 12 a 35]* e subito dopo invece di far eseguire le istruzioni gli mettiamo un'altro enunciato *For colonna = 7 To 18* in questo caso il contatore del ciclo è rappresentato dalla variabile colonna e lo interpretiamo così *[Per colonna che và da 7 a 18]* , a questo punto abbiamo posto due condizioni una sotto l'altra e subito dopo abbiamo posto le istruzioni *Cells(riga, colonna).Value = riga* in questo enunciato la parola chiave *Cells* (che vedremmo nella prossima lezione) sta ad indicare una determinata cella del nostro foglio localizzata dai valori delle variabili riga e colonna, l'altra parola chiave *Value* indica il valore da inserire nelle coordinate rappresentate, tale valore lo abbiamo identificato con riga.

Seguendo l'esecuzione del ciclo, prima viene eseguito il ciclo interno, e cominciamo dalla cella che si trova all'intersezione tra la colonna n° 7 e la riga n° 12che sul nostro foglio è rappresentata da **G12**, al cui interno scriviamo il valore di riga (per cui 12), poi incontriamo la parola chiave *Next colonna*, a questo punto però non abbiamo ancora raggiunto la determinante del ciclo (rappresentato dal valore [18]) e VBA incrementa il contatore di 1 e ripete le istruzioni, così facendo andiamo a scorrere tutte le colonne e scriviamo al loro interno il valore della variabile riga.

Una volta raggiunta la determinante del ciclo usciamo dal ciclo interno ma troviamo la parola chiave *Next riga*, per cui il contatore del ciclo esterno viene incrementato, passiamo alla riga successiva (la 13) e ripetiamo il ciclo interno come abbiamo descritto sopra. Il nostro listato ha fine quando viene raggiunta la determinate del ciclo esterno e a questo punto troveremo la nostra area di lavoro riempita con il valore della variabile riga estesa sulle colonne di ciascuna riga. Ora che abbiamo riempito l'area con il valore della riga tramite un ciclo For Next nidificato potremmo vedere un listato Do Loop per fare il contrario, cioè riempire l'area col valore della colonna.

```
Sub scrivi_col_dolop()
riga = 12: colonna = 7
Do Until riga = 35
 Do While colonna <> 19
 Cells(riga, colonna).Value = colonna
 colonna = colonna + 1
 Loop
riga = riga + 1: colonna = 7
Loop
End Sub
```

Notiamo subito una netta differenza tra i due cicli, infatti abbiamo dovuto dichiarare i valori iniziali delle variabili riga e colonna prima dell'inizio del ciclo*(riga = 12: colonna = 7)* in questa tipologia ciclica il VBA non riesce a determinare da dove deve iniziare, in quanto la sintassi di espressione è diversa, una volta indicati i valori di partenza troviamo la prima parola chiave *Do Until riga = 35* [ripeti finchè il valore di riga non è uguale a 35] e subito dopo abbiamo posto l'altra condizione che costituisce il ciclo interno *Do While colonna <> 19* facciamo attenzione a questo enunciato, abbiamo usato la parola chiave While e il codice viene interpretato così[ripeti mentre colonna è diversa da 19] ma perchè 19 dato che l'ultima colonna che dobbiamo riempire è la n° [18]?.

Lo comprendiamo subito andando avanti con l'analisi del ciclo, possiamo saltare la spiegazione delle istruzioni *Cells(riga, colonna).Value = colonna* in quanto la loro funzione è uguale a quanto spiegato sopra per il ciclo For con la sola differenza che riempiamo le celle col valore della variabile colonna, mentre la soluzione al quesito posto sta nella riga sotto *colonna = colonna + 1* questo è il nostro contatore, il metodo che usa *Do ... Loop* per incrementarlo e continuare nella sua esecuzione.

Infatti alla prima esecuzione colonna vale 7 e viene incrementata sempre di 1, poi quando trova la parola chiave *Loop* ritorna all'altra parola chiave *Do* (dove ha iniziato il ciclo) ed esegue ancora le istruzioni, ma nella determinante del ciclo abbiamo messo 19 (un valore in più di [18]) semplicemente perchè quando incrementiamo il valore della variabile colonna con la dicitura *colonna = colonna + 1* alla parola chiave *Loop la determinate del ciclo sarebbe soddisfatta se mettessimo [18]* e di conseguenza la nostra area verrebbe riempita fino alla colonna 17

Provate a modificare il valore nell'esempio allegato e comprenderete come agisce il ciclo, inoltre quando si lavora con i cicli e per un motivo qualsiasi ci viene rimandato un errore oppure non viene eseguito quello che volevamo, possiamo usare un piccolo espediente per vedere come agisce il ciclo e quale valore attribuisce alle variabili usate, infatti basta usare la parola chiave Msgbox e possiamo vedere a video il valore della variabile e correggere l'errore. Ecco un esempio di utilizzo

```
Sub scrivi_col_dolop()
riga = 12: colonna = 7
MsgBox riga
Do Until riga = 35
Do While colonna <> 19
MsgBox colonna
Cells(riga, colonna).Value = colonna
colonna = colonna + 1
Loop
riga = riga + 1: colonna = 7
Loop
End Sub
```

L'Istruzione Loop Wend

L'istruzione Loop Wend ripetere un'azione finché una condizione è vera che può essere interpretata come:

While condizione da verificare
azione intrapresa
Wend

Se la condizione è vera, vengono eseguite le azioni indicate nella procedura e quando viene raggiunta l'istruzione Wend, la procedura ritorna all'operazione While e la condizione viene verificata nuovamente. Se la condizione è ancora vera, il processo viene ripetuto, mentre invece se la condizione è falsa, l'esecuzione salta alla prima riga che di codice che segue l'istruzione Wend .

```
Sub Test1()
   Dim i As Integer
   i = 1
   'Loop attraverso le celle della colonna A
    'Si esce dal ciclo se la cella (Cells (i, 1)) è vuota
   While Not IsEmpty(Cells(i, 1))
      'Scrivere il contenuto della cella nella finestra di esecuzione.
      Debug.Print Cells(i, 1)
      'Incrementa la variabile di una unità per testare la cella successiva.
      i = i + 1
   Wend
End Sub
```

L'istruzione Exit Do

È possibile uscire dal ciclo Do While senza completare il ciclo completo, utilizzando la dichiarazione **Exit Do** che arresta immediatamente l'esecuzione del ciclo ed esegue la sezione di codice immediatamente successiva all'istruzione Loop, e nel caso di cicli nidificati si puà uscire dal ciclo interno ed eseguire il successivo livello esterno, in quanto si può avere qualsiasi numero di istruzioni Exit Do in un ciclo. Questa istruzione è particolarmente utile nel caso in cui si desidera terminare il ciclo al raggiungimento di un certo valore o di soddisfare una condizione specifica, o nel caso in cui si desidera interrompere un loop infinito a un certo punto.

Esempio: Se la cella A1 è vuota, *nTotal* si sommano al valore di 55. Se Range ("A1") contiene il valore 5, il ciclo termina e uscire quando il contatore (es. n) raggiunge i 5 e nTotal si sommano a 10 (si noti che il ciclo non viene eseguito per il controvalore di 5, ed esce il ciclo al raggiungimento di questo valore).

```
Sub Test2 ()
Dim n As Integer , nTotal As Integer
nTotal = 0
Do While n <11
nTotal = n + nTotal
n = n + 1
If n = ActiveSheet.Range ("A1") Then
Exit Do
End If
Loop
MsgBox nTotal
End Sub
```

Forzare l'uscita dal ciclo

Forzare l'uscita di un ciclo si corre sempre il rischio di creare un ciclo infinito se la condizione di uscita non viene mai soddisfatta e in caso di emergenza, se una macro crea un Loop infinito si può fermare la sua esecuzione premendo contemporaneamente i tasti della **Ctrl + Pausa** . È anche possibile premere il tasto Esc e in questo caso viene visualizzata una finestra di errore di esecuzione interrotta, in cui si deve cliccare sul tasto Fine per completare la procedura. Se si desidera gestire i tasti Ctrl + Pausa o Esc in una macro, si deve utilizzare la proprietà *EnableCancelKey*: L'esempio sotto riportato mostra come personalizzare la finestra del messaggio di errore

```
Sub Test3()
    Dim x As Long
    On Error GoTo Fine
    Application.EnableCancelKey = xlErrorHandler
     'si crea un loop per dare tempo di testare la procedura
    'dell'uso del tasto Esc
    For x = 1 To 50000
        Cells(x, 1) = x
    Next x
Fine:
   If Err.Number = 18 Then MsgBox "Operazione Annullata"
End Sub
```

Azioni ripetitive: Il Ciclo For e il ciclo For Each

Ora che sappiamo come scegliere diverse azioni basandosi su delle condizioni predeterminate, siamo pronti a vedere come ripetere delle azioni un numero predeterminato di volte se si verifica, o non si verifica una particolare condizione nella nostra procedura. Uno degli svantaggi delle macro è la loro incapacità di ripetere le azioni a meno che le azioni desiderate non vengano registrate ripetutamente, VBA fornisce diverse strutture potenti e versatili per permetterci di ripetere facilmente le azioni e per controllare il modo in cui VBA effettua queste ripetizioni.

Le strutture del programma che ripetono l'esecuzione di una o più istruzioni sono chiamate **Cicli**, alcune strutture per i cicli sono costruite in modo da venire eseguite un numero impostato di volte, e vengono chiamate *cicli ad interazione fissa*, mentre altri tipi di strutture per i cicli vengono impostate un numero variabile di volte sulla base di alcune condizioni impostate, proprio perchè il numero di ripetizioni di queste strutture non è definito questi cicli vengono chiamati *cicli ad interazione indefinita*.

Sia nelle strutture ad interazione fissa che nelle strutture ad interazione indefinita, ci sono alcune espressioni che determinano quante volte il ciclo viene ripetuto, questa espressione viene chiamata *determinante del ciclo*, questa espressione in un ciclo ad interazione fissa è quasi sempre una espressione numerica, mentre per i cicli a interazione indefinita la determinante del ciclo è solitamente un'espressione logica che descrive la condizione sotto la quale il ciclo può continuare o interrompere la sua esecuzione, praticamente vengono usate delle espressioni logiche per la determinante dei cicli allo stesso modo in cui sono state usate per prendere delle decisioni in VBA.

Il ciclo For Next

La più semplice struttura per i cicli è quella ad interazione fissa, VBA ne fornisce due diverse tipologie che vengono espresse così For ...Next e For ...Each ... Next entrambi i cicli vengono chiamati *cicli For* perchè vengono eseguiti per uno specifico numero di volte. Il ciclo For ... Next ha la seguente sintassi

For contatore = inizio To fine
istruzioni
Next contatore

contatore viene rappresentato da una qualsiasi variabile numerica, di tipo *Integer* o *Long*, *inizio* è anch'esso rappresentato da una variabile numerica e specifica il valore iniziale per la variabile *contatore*, anche *fine* è una espressione numerica che rappresenta il valore finale per la variabile *contatore*. Per default VBA incrementa la variabile [I]contatore di **1** ogni volta che esegue le istruzioni di un ciclo. La parola chiave **Next** indica a VBA che è stata raggiunta la fine del ciclo, inoltre la variabile *contatore*, dopo la parola chiave *Next*, deve essere la stessa variabile *contatore* che abbiamo messo appena dopo l'enunciato For all'inizio della struttura del ciclo.

Quando VBA esegue un ciclo For, prima assegna il valore rappresentato da inizio alla variabile *contatore*, quindi esegue tutte le istruzioni rappresentate da *istruzioni* fino a quando non raggiunge la parola chiave *Next* che indica che è stata raggiunta la fine del ciclo. VBA poi incrementa la variabile *contatore* di 1 e ritorna all'inizio del ciclo e confronta il valore corrente della variabile *contatore* col valore rappresentato da *fine*. Se *contatore* è minore o uguale a *fine* esegue nuovamente il ciclo. Se invece *contatore* è maggiore di *fine* VBA esce dal ciclo e continua la sua esecuzione con la prima istruzione dopo la parola chiave *Next*.

Possiamo però anche specificare un valore diverso per l'incremento della variabile contatore includendo la parola chiave **Step** [opzionale], in questo caso dobbiamo specificare l'incremento della variabile *contatore*, e la sintassi diventa così:

For contatore = inizio To fine Step passo
istruzioni
Next contatore

In questo caso l'espressione *passo* viene rappresentato da una espressione numerica e indica la quantità per incrementare la variabile *contatore*, vediamo qualche esempio

```
Sub for_semplice()
 For I = 1 To 10
 MsgBox (I)
 Next I
End Sub
```

L'esecuzione di questo codice ci porta a video la finestra di messaggio (MsgBox) per 10 volte con il valore della variabile I (*contatore*), se invece vogliamo usare la parola chiave *Step* e far apparire solo le espressioni dispari possiamo modificare il listato in questo modo

```
Sub for_step()
 For I = 1 To 10 Step 2
 MsgBox (I)
 Next I
End Sub
```

E ci verranno riportati i soli valori dispari della variabile nella finestra di dialogo di MsgBox. Con la parola chiave *Step*, abbiamo modificato l'incremento della variabile da **1** (di default) a **2**, in pratica alla prima esecuzione del ciclo **I** *vale 1* ma quando incontra la parola chiave *Step* viene incrementata di **2**, se fate "girare" la macro for_step noterete che al primo ciclo viene stampato il valore 1, questo perchè **I** quando incontra la parola chiave *Step* vale ancora 1, per cambiare valore deve arrivare alla parola chiave *Next* (che abbiamo detto essere quella che avvisa di essere arrivati alla fine del ciclo e che incrementa la variabile).

In ultima analisi l'enunciato For viene interpretato così : *For I* [Per I] = *1 To 10* [che và da 1 a 10] *Step 2* [incrementa di 2], *Esegui le istruzioni , Next I*[incrementa il valore di I], Il ciclo *For ..Next* ha la flessibilità di poter incrementare e decrementare la variabile, possiamo modificare il listato da *For I = 1 To 10 Step 2* a *For I = 100 To 1 Step -2* pertanto le possibilità di impiego sono abbastanza vaste, possiamo eseguire somme, incrementare e decrementare il valore delle variabili, utilizzare la ciclicità per ogni bisogno che richieda il nostro programma fermo restando i principi di impiego esposti all'inizio. Esiste anche un'altra forma di ciclo ed è il ciclo *For Each ..Next*

Il ciclo For Each

A differenza del ciclo *For … Next* il ciclo **For Each ... Next** non usa un contatore, ma viene eseguito tante volte quanti sono gli elementi di un gruppo specifico, come una collezione di oggetti o un vettore. In breve il ciclo For Each … Next viene eseguito una volta per ogni elemento di un gruppo. Questo tipo di ciclo ha la seguente sintassi

For Each elemento In gruppo
Istruzioni
Next [elemento]

Dove, *elemento* è una variabile usata per interare il ciclo per tutti gli elementi del gruppo, *gruppo* è una collezione di oggetti o un vettore (matrice), se *gruppo* è una collezione di oggetti, *elemento* deve essere una variabile di tipo *Variant* o *Object* o uno specifico tipo di oggetto come *Range*, *Worksheet*, *Cells*e così via. *Istruzioni* rappresenta nessuna, una o più istruzioni VBA che formano il corpo del ciclo. Questo tipo di ciclo ha meno opzioni del ciclo For Next, l'incremento del contatore non è rilevante in questo ciclo, perchè viene sempre eseguito tante volte quanti sono gli elementi presenti nel gruppo specificato. In pratica non dovendo indicare il numero iniziale e finale per impostare quante volte il ciclo deve ripetere le nostre istruzioni, la nostra ricerca sarà rappresentata da quante volte il dato da cercare sarà presente nell'area di ricerca e l'inizio sarà sempre dal primo record contenuto nell'area di ricerca.

Per meglio comprendere vediamo un esempio di questo tipo : Supponiamo di avere un elenco di nomi nell'intervallo di celle (Range) A1:A5 e si deve confrontare il valore della cella B1 e verificare se è presente nel Range, possiamo scrivere questo codice :

```
Sub prova_each()
Dim x As Boolean
y = Range("B1").Value
  For Each cl In Range("A1:A5")
   If cl = y Then
   x = True
  End If
 Next
If x = True Then
  MsgBox ("Valore Presente")
Else
  MsgBox ("Valore Assente")
End If
End Sub
```

In questo listato *elemento* è rappresentato dalla variabile **y**, cioè dal valore contenuto nella cella B1 e *gruppo* è rappresentato dal Range A1:A5, mentre il corpo del ciclo è rappresentato dalle istruzioni MsgBox se il valore è presente oppure no. In questo tipo di ciclo il numero iniziale da cui partire è definito dalla prima cella del Range cioè A1 e il numero finale dall'ultima cella dello stesso Range, cioè A5. Adesso il ciclo sa dove iniziare, dove finire e cosa cercare (cerca il valore della cella B1) e scorrendo tutte le celle del Range esegue le istruzioni MsgBox se il valore di B1 è presente oppure no nel percorso di confronto Possiamo quindi semplificare affermando che gruppo può essere un insieme di oggetti come le celle, le righe e le colonne di un foglio di calcolo, i fogli di lavoro di un file Excel oppure gli oggetti che possiamo inserire in un foglio come pulsanti, caselle di testo pulsanti di comando Userform etc..

E' indubbio che un ciclo For Each rappresenta un modo pratico per sfogliare gli insiemi di una collezione, va ricordato che l'istruzione For Each funziona come Set, ovvero assegna un riferimento dell'oggetto a una variabile, ma invece di assegnarne uno solo, sceglie tutti gli elementi di una raccolta. Quindi, per ogni oggetto della raccolta, Visual Basic esegue tutte le istruzioni fino all'istruzione Next, anche se tecnicamente, non è necessario inserire il nome della variabile dopo Next, se lo si utilizza, Visual Basic richiede che corrisponda al nome della variabile dopo For Each. Si consiglia di utilizzare sempre la variabile del ciclo dopo Next in modo che Visual Basic possa contribuire a evitare bug nella macro. Le istruzioni che hanno inizio con For Each e terminano con Next sono definite blocchi For Each o cicli For Each.

Cicli Nidificati

È possibile nidificare cicli mettendo un ciclo all'interno di un altro ciclo, fino a un numero illimitato di volte. La variabile contatore per ogni ciclo deve essere univoca ed è possibile nidificare un tipo di ciclo all'interno di un altro tipo diverso. In un ciclo for, è necessario che il ciclo interno sia completato prima che si esegua l'istruzione successiva del ciclo esterno. È inoltre possibile nidificare un tipo di struttura di controllo all'interno di un altro tipo vale a dire che è possibile nidificare un'istruzione IF all'interno di un blocco With che può a sua volta essere annidato all'interno di un For ... Each. Tuttavia, le strutture di controllo non possono essere sovrapposte, ogni blocco nidificato si deve chiudere e terminare entro il livello nidificato esterno.

Esempio di nidificazione di un ciclo IF all'interno di un blocco With che è annidato all'interno di un For. Il codice del ciclo Loop scorre ogni cella nell'intervallo A1: A10, e se il valore della cella è superiore a 5, il colore di sfondo viene impostato come giallo, mentre per valori inferiori a 5 viene usato il colore rosso

```
Sub ciclo1()
Dim iCell As Range
For Each iCell In ActiveSheet.Range("A1:A10")
With iCell
If iCell > 5 Then
.Interior.Color = RGB(255, 255, 0)
Else
.Interior.Color = RGB(255, 0, 0)
End If
End With
Next iCell
End Sub
```

Uscita anticipata dal ciclo

È possibile uscire da un ciclo For (For ... Next e For Each ... Next) in anticipo, senza completare il ciclo, utilizzando la dichiarazione Exit For che interromperà immediatamente l'esecuzione del ciclo esistente e passerà ad eseguire la sezione di codice immediatamente successiva all'istruzione Next, e nel caso di livello nidificato interno si ferma ed eseguirà il successivo livello nidificato esterno. Si può avere qualsiasi numero di istruzioni Exit For in un ciclo ed è particolarmente utile nel caso in cui si desidera terminare il ciclo al raggiungimento di un certo valore o di soddisfare una condizione specifica, o nel caso in cui si desidera interrompere un Loop infinito a un certo punto.

Esempio: Se la cella A1 è vuota, la variabile nTotal si somma al valore 25, mentre se la cella A1 contiene il valore 5, il ciclo termina ed esce quando il contatore raggiunge il valore 5

```
Sub ciclo2 ()
Dim n As Integer, nTotal As Integer
nTotal = 0
For n = 1 To 10
nTotal = n + nTotal
If n = ActiveSheet.Range("A1") Then
Exit For
End If
Next n
MsgBox nTotal
End Sub
```

Cartelle di lavoro Nozioni di base

Quando si avvia Excel, viene creata di default una cartella di lavoro vuota e dopo aver eseguito le varie elaborazioni è possibile salvarla, oppure, se si dispone di una cartella di lavoro esistente da qualche parte nel computer è possibile aprirla come un documento di Excel. Nel linguaggio VBA, una cartella di lavoro è un oggetto che appartiene ad una collezione chiamata *cartelle di lavoro* e ogni cartella di lavoro è un oggetto di tipo **Workbook**.

Per quanto riguarda le collezioni, ogni cartella di lavoro può essere identificata utilizzando le sue proprietà e per fare riferimento a una cartella di lavoro in programmazione, si deve accedere alle sue proprietà usando l'indice della cartella o il nome della cartella di lavoro stessa. Dopo aver fatto riferimento a una cartella di lavoro, se si desidera eseguire un'azione, è necessario ottenere un riferimento dichiarando una variabile e assegnarle una chiamata che dovrebbe essere fatta nel modo seguente.

```
Sub Prova1()
   Dim cart As Workbook
   Set cart = Workbooks.Item(2)
End Sub
```

Come già detto, una cartella di lavoro è un oggetto di tipo *Workbook* che per sostenere la possibilità di creare una nuova cartella, la raccolta cartelle di lavoro è dotata di un metodo denominato **Add**. La sua sintassi è.

Workbooks.Add(Template) As Workbook

Questo metodo richiede un solo argomento, ed è facoltativo, ciò significa che è possibile richiamare il metodo senza argomenti e senza parentesi, ecco un esempio.

```
Sub Prova1 ()
    Workbooks.Add
End Sub
```

Quando il metodo viene richiamato in questo modo, viene creata una nuova cartella di lavoro e per fare questo, si deve tenere presente che il metodo **Add ()** restituisce un oggetto Cartella di lavoro, pertanto, quando si crea una cartella, si ottiene anche un riferimento ad essa. Si ottiene tutto ciò assegnando la chiamata al metodo per la creazione di una cartella di lavoro ad una variabile, ecco un esempio.

```
Sub Prova1 ()
   Dim cart As Workbook
   Set cart = Workbooks.Add
End Sub
```

Con questo codice è possibile quindi utilizzare la variabile per modificare le proprietà della cartella di lavoro. Dopo aver lavorato su una nuova cartella di lavoro, o dopo averla creata a livello di codice, se si desidera mantenere la cartella quando l'utente chiude Excel, è necessario salvarla e l'utente ha la possibilità di utilizzare la finestra di dialogo Salva. Quando l'utente avvia il salvataggio di un file, viene visualizzata la finestra di dialogo Salva con nome, che per default mostra il contenuto della cartella Documenti. Per scoprire quale sia la cartella di default, è possibile fare clic sul pulsante Opzioni di Excel e nella finestra di dialogo Opzioni di Excel, verificare il contenuto del file predefinito nella casella di testo Percorso.

Per modificare a livello di programmazione la cartella predefinita, la classe Application è dotato di una proprietà denominata *DefaultFilePath*, pertanto, per specificare a livello di codice la cartella predefinita, si deve assegnare una stringa alla proprietà **Application.DefaultFilePath**, ecco un esempio.

```
Sub Prova1 ()
    Application.DefaultFilePath = "C: \Prova\gestione preventivi"
End Sub
```

Quando questo codice viene eseguito, il file predefinito della finestra di dialogo Opzioni di Excel verrà cambiato. Per salvare visivamente una cartella di lavoro, è possibile fare clic sul pulsante ***Salva***, oppure è anche possibile premere i tasti ***Ctrl + S***, mentre se il documento è già stato salvato, l'operazione verrebbe eseguita senza nessun consenso da parte dell'utente. Per poter avere la possibilità di salvare una cartella di lavoro a livello di programmazione, la classe cartella di lavoro è dotata di un metodo denominato **Save**, la sua sintassi è.

Workbook.Save ()

Come si può vedere, questo metodo non richiede nessun argomento, se si clicca sul pulsante Salva o se si richiama il metodo *Workbook.Save ()* su una cartella che non è stata ancora salvata, verrebbe richiesto di fornire un nome alla cartella di lavoro. Per salvare una cartella di lavoro in una posizione diversa, è possibile fare cliccare sul pulsante Salva con nome e selezionare una delle opzioni presentate, oppure è anche possibile premere F12. Come ausilio alla programmazione per il salvataggio di una cartella di lavoro, la classe cartella di lavoro è dotata di un metodo denominato Salva con nome. La sua sintassi è.

Workbook.SaveAs(FileName, FileFormat, Password, WriteResPassword, ReadOnlyRecommended, CreateBackup, AccessMode, ConflictResolution, AddToMru, TextCodepage, TextVisualLayout, Local)

Il primo argomento è l'unico richiesto e contiene il nome o il percorso del file, pertanto, è possibile fornire solo il nome del file con estensione quando si richiama. Ecco un esempio.

```
Sub Prova1 ()
        Dim cart As Workbook
   Set cart = Workbooks.Add cart.SaveAs "cart.xlsx"
End Sub
```

Se si fornisce solo il nome di un file quando si richiama questo metodo, la nuova cartella di lavoro sarebbe salvata nella directory corrente o nella cartella Documenti, mentre se si desidera, un'alternativa si deve fornire il percorso completo del file.

Apertura di una cartella di lavoro

Excel è un'interfaccia a documenti multipli (MDI), ciò significa che è possibile aprire molte cartelle di lavoro allo stesso tempo, per questo motivo, la possibilità di aprire una cartella di lavoro a livello di programmazione è gestita dalla raccolta cartelle di lavoro e a sostegno di questo, la classe è dotata di un metodo denominato **Open**, la sua sintassi è.

Workbooks.Open(FileName, UpdateLinks, ReadOnly, Format, Password, WriteResPassword, IgnoreReadOnlyRecommended, Origin, Delimiter, Editable, Notify, Converter, AddToMru, Local, CorruptLoad)

FileName è l'unico argomento obbligatorio e quando si richiama questo metodo, è necessario fornire il nome del file o il suo percorso, ciò significa che è possibile fornire un nome di file con la sua estensione, ecco un esempio.

```
Sub Prova1 ()
    Workbooks.Open "cart.xlsx"
End Sub
```

Se si fornisce solo il nome del un file, Excel avrebbe cercato nella directory corrente o nella cartella Documenti, se invece Excel non è in grado di archiviare il file, come si può immaginare, si riceverà un errore, mentre una sicura alternativa è quella di fornire il percorso completo del file.

Chiusura cartelle di lavoro

Dopo aver usato una cartella di lavoro o per chiudere un documento non più necessario, la classe cartella di lavoro è dotata di un metodo denominato **Close**, la sua sintassi è.

Sub Close (Optional ByVal SaveChanges As Boolean, Optional ByVal Filename As String, Optional ByVal RouteWorkbook As Boolean)

Tutti e tre gli argomenti sono opzionali, il primo argomento indica se si desidera salvare le modifiche, se sono state fatte, da quando è stata aperta la cartella di lavoro. Se non è stato fatto nessun cambiamento dal momento in cui la cartella di lavoro è stata creata o dall'ultima volta che è stata aperta, questo argomento non viene considerato. Se il primo argomento è impostato su True e la cartella di lavoro ha delle modifiche che devono essere salvare, il secondo argomento specifica il nome del file in cui salvare la cartella di lavoro, mentre invece il terzo argomento specifica se la cartella di lavoro deve essere disponibile per l'utente successivo.

Accesso a una cartella di lavoro

Per accedere a una cartella di lavoro, la classe cartella di lavoro è dotata di un metodo denominato **Activate** e la sua sintassi è.

Workbook.Activate ()

Questo metodo non richiede alcun argomento, pertanto, per richiamarlo, è possibile ottenere un riferimento alla cartella di lavoro che si desidera accedere, quindi si deve richiamare il metodo Activate () in questo modo.

```
Sub Prova1 ()
Dim cart As Workbook
   Set cart = Workbooks.Item (2) cart.Activate
End Sub
```

È anche possibile usare meno codice applicando l'indice direttamente alla cartelle di lavoro. Ecco un esempio.

```
Sub Prova1 ()
    Workbooks (2).Activate
End Sub
```

Visualizza Molte cartelle di lavoro

Se si crea o si aprono molte cartelle di lavoro, ognuna è rappresentata sulla barra delle applicazioni da un pulsante. Per fare riferimento a una cartella di lavoro, si deve accedere alle proprietà e passare l'indice o il nome del file della cartella di lavoro. Ecco un esempio:

```
Sub Prova1 ()
   Workbooks.Item (2)
End Sub
```

Dopo aver fatto riferimento a una cartella di lavoro, se si desidera eseguire un'azione, è necessario ottenere un riferimento e per fare questo, si deve dichiarare una variabile cartella di lavoro e assegnare la chiamata ad essa. Ciò dovrebbe essere fatto nel modo seguente:

```
Sub Prova1 ()
Dim cart As Workbook
   Set cart = Workbooks.Item (2)
End Sub
```

Classi e Oggetti: Introduzione

Il linguaggio di Microsoft Visual Basic utilizza il concetto di classe per identificare o gestire le parti di un'applicazione. Se, per esempio, consideriamo un oggetto come una casa, avrà delle caratteristiche come il tipo (residenziale, condominio, villa, etc.), il numero di camere da letto, il numero di bagni, etc. e tutte queste caratteristiche le utilizziamo per descrivere la casa a qualcuno che vuole acquistarla. In programmazione, per ottenere un tale oggetto, è necessario definire i criteri che lo descrivono.
Ecco un esempio:

House
[indirizzo
tipo_di_casa
numero_di_camere
numero_di_bagni
ha_il_garage]

Queste informazioni vengono utilizzate per descrivere la casa e sulla base di questo, la casa (**House**) si chiama classe. In realtà per descrivere una vera e propria casa, è necessario fornire informazioni più dettagliate per ciascuna delle caratteristiche di cui sopra, come per esempio:

House: Mandela
indirizzo: V. Libertà 123
tipo_di_casa: Monofamiliare
numero_di_camere: 4
numero_di_bagni: 3
ha_il_garage: Sì

In questo caso, la casa di nome **Mandela** non è più una classe, ma è una casa vera e propria esplicitamente descritta, pertanto, Mandela è un oggetto. Sulla base di questo, possiamo definire una classe come una tecnica utilizzata per fornire i criteri per definire un oggetto, che è il risultato di una descrizione basata su una classe.

Le proprietà di un oggetto

Nel nostro esempio della casa, abbiamo usato dei termini per descriverla, come: Indirizzo, Tipo di casa, numero di camere, numero di bagni etc. in programmazione invece le caratteristiche utilizzate per descrivere un oggetto sono indicate come sue **proprietà**. Mentre la maggior parte degli oggetti forniscono solo le caratteristiche per descriverli, altri possono eseguire azioni, ad esempio, una casa può essere utilizzata per proteggere le persone quando fuori piove. In programmazione, invece, un'azione che un oggetto può eseguire è indicato come **metodo**. In precedenza, abbiamo definito una classe **Casa**con le sue proprietà, ma a differenza di una proprietà, un metodo deve visualizzare delle parentesi sul lato destro per distinguerlo da una proprietà. Un esempio potrebbe essere:

House
[indirizzo
tipo_di_casa
numero_di_camere
numero_di_bagni
ha_il_garage
Proteggi_pioggia ()]

Quando un oggetto ha un **metodo**, per accedere a tale metodo, si deve immettere il nome dell'oggetto, seguito da un punto, e dal nome del metodo con le parentesi.

Ad esempio, se si dispone di un oggetto casa di nome Mandela e volete chiedere di proteggere dalla pioggia, si deve digitare:

Mandela.Proteggi_pioggia ()

Questo è indicato anche come *richiamare un metodo*.

Quando è stato chiesto di eseguire un'azione, un metodo può avere bisogno di uno o più valori con cui lavorare, se un metodo necessita di un valore, tale valore è chiamato **argomento**. Mentre un certo metodo può avere bisogno di un argomento, un altro metodo potrebbe averne bisogno più di uno e il numero di argomenti di un metodo dipendono dalla sua portata e sono visualizzati tra parentesi. Supponiamo di avere un oggetto casa e si vuole proteggere il suo contenuto, per svariate ragioni, per cui l'interno deve essere protetto per:

- La pioggia
- La polvere
- Il vento
- Il sole, etc.

Sulla base di questo, potrebbe essere necessario fornire informazioni aggiuntive per indicare perché o come la parte interna deve essere protetta. Per questo motivo, quando tale metodo viene chiamato, queste ulteriori informazioni devono essere fornite, tra le parentesi del metodo. Ecco un esempio:

House
[indirizzo
tipo_di_casa
numero_di_camere
numero_di_bagni
ha_il_garage
Proteggi_pioggia (ragione)]

Come accennato in precedenza, un metodo può essere creato per prendere più di un argomento. In questo caso, gli argomenti sono separati da virgole. Ecco un esempio:

House
[indirizzo
tipo_di_casa
numero_di_camere
numero_di_bagni
ha_il_garage
Proteggi_pioggia (ragione, quando)]

Gli argomenti vengono utilizzati per aiutare l'oggetto ad eseguire l'azione prevista e una volta creato il metodo, può essere utilizzato varie volte e se un metodo necessita di un argomento, quando si richiama, è necessario fornire un valore, altrimenti il metodo non funzionerebbe. Per richiamare un metodo che richiede un argomento, si deve digitare il nome del metodo seguito dalla parentesi aperta "**(**" e seguito dal valore che sarà l'argomento e seguito da una parentesi chiusa "**)**". L'argomento che si passa può essere un valore costante, regolare o può essere il nome di un altro oggetto e se il metodo richiede più di un argomento, per richiamarlo, si devono digitare i valori per gli argomenti, nell'ordine esatto indicato, separati l'uno dall'altro da una virgola.

Abbiamo detto che, quando si richiama un metodo che richiede un argomento, è necessario fornire un valore per l'argomento, ma c'è un'eccezione.

A seconda di come è stato creato il metodo, può essere configurato per utilizzare un valore di default se non si fornisce un valore, ma non tutti i metodi seguono questa regola.
Se un metodo che accetta un argomento ha un valore di default per questo, allora non c'è bisogno di fornire un valore quando si chiama questo metodo e tale argomento è considerato facoltativo, inoltre gli argomenti che hanno valori predefiniti possono essere utilizzati e non c'è bisogno di fornirli.

Tecniche di Accesso ad un oggetto: L'enunciato Me

Finora abbiamo visto che un oggetto dispone di proprietà e metodi e abbiamo anche visto come accedere a una proprietà di un oggetto. Per esempio, immaginate di avere una classe House definita come segue:

House
[indirizzo
tipo_di_casa
numero_di_camere
numero_di_bagni
ha_il_garage
Proteggi_pioggia ()]

Se si dispone di un oggetto denominato **Gino** e che è di tipo **House**, per accedere ad alcune delle sue proprietà, è necessario utilizzare il codice come segue:

```
Gino.indirizzo
Gino.tipodicasa
```

Se si sta lavorando all'interno di un metodo della classe, per esempio, se si lavora nel corpo del metodo Proteggi_pioggia, è anche possibile accedere alle proprietà nello stesso modo, questa volta senza il nome dell'oggetto. Ciò potrebbe essere fatto nel modo seguente:

```
Proteggi_pioggia ()
indirizzo
tipo_di_casa
numero_di_camere
numero_di_bagni
End
```

Quando si accede ad un membro di una classe all'interno di uno dei suoi metodi, è possibile precedere il membro con l'oggetto **Me** preceduto dall'operatore (punto). Ecco un esempio:

```
Proteggi_pioggia ()
Me. indirizzo
Me. Tipo_di_casa
Me. Numero_di_camere
Me. Numero_di_bagni
End
```

Ricordate che l'oggetto **Me** viene utilizzato per accedere ai membri di un oggetto mentre si è all'interno di un altro membro dell'oggetto.

Il ciclo With

Abbiamo visto che è possibile utilizzare il nome di un oggetto per accedere ai suoi membri. Ecco un esempio:

```
Gino. indirizzo
Gino. tipo_di_casa
Gino. numero_di_camere
Gino. Numero_di_bagni
```

Invece di utilizzare il nome dell'oggetto ogni volta, è possibile avviare una sezione con la parola chiave **With** seguita dal nome dell'oggetto con espressione:

```
With Gino

End With
```

Tra le parole chiave **With** e **End With** si inseriscono le linee di codice per accedere al membro della classe preceduto da un punto seguito dal membro desiderato. Ciò dovrebbe essere fatto nel modo seguente:

```
With Gino
.indirizzo
.tipo_di_casa
.numero_di_camere
.numero_di_bagni
.ha_il_garage
End With
```

Classi, oggetti ed eventi personalizzati

In VBA è possibile creare oggetti personalizzati attraverso la definizione di classi e l'inserimento di moduli di classe. È inoltre possibile creare eventi di classe e procedure di evento personalizzati integrati in Excel. Con questa premessa risulta abbastanza chiaro che tramite VBA è possibile definire e creare delle classi di oggetti che possano soddisfare le esigenze del singolo utente. Per definizione la programmazione orientata agli oggetti prevede di raggruppare in un'unica entità, ovvero in una classe, sia la struttura dati che le procedure che operano su di essa, creando un oggetto unico dotato di proprietà e metodi che operano sull'oggetto stesso.

La classe è un modello per i nuovi oggetti che verranno creati e sono utilizzate per memorizzare, elaborare e rendere disponibili i dati, in quanto oltre ai dati contengono anche il codice per gestirli con procedure e metodi come le *Sub* e le *Function*. Una classe viene creata inserendo un modulo di classe nel progetto VBA e consente di creare i propri oggetti con proprietà e metodi molto simili ad altri oggetti come Range, foglio, grafico, etc. Il modulo di classe ha una serie di procedure che includono variabili e costanti che definiscono le sue proprietà che possono essere manipolate in un modulo di classe con le procedure *Property Let*, *Property Get* e *Property Set*. Per accedere alle proprietà e metodi dell'oggetto classe da una routine in un modulo di codice, si dichiara una variabile oggetto del tipo della classe.

Si può programmare in VBA, senza creare oggetti personalizzati che in realtà non aumentano la funzionalità del codice, tuttavia, l'utilizzo di oggetti personalizzati rende la codifica meno complessa e più semplice avendo un collegamento nel codice rendendo la codifica auto documentata denominando le classi in modo appropriato, e questo aiuta il debug e il codice riutilizzo.

Inserire un modulo di classe

In Visual Basic Editor (VBE), per inserire un modulo di classe si deve cliccare su **Inserisci - Modulo di classe**, in alternativa, nella finestra dei *progetti*, cliccare con il pulsante destro del mouse sul nome del progetto e sul menu che appare cliccare su *Inserisci* e quindi scegliere *Modulo di classe*. In alternativa, sulla barra degli strumenti standard in VBE, cliccare sul pulsante *Inserisci* e quindi scegliere *Modulo di classe*. Questo crea una classe vuota con il nome di *Class1*. Per rimuovere o eliminare un modulo di classe, si deve cliccare con il tasto destro del mouse dopo aver selezionato il modulo nella finestra dei progetti e quindi fare clic su *Rimuovi*

Le istanze di proprietà di un modulo di classe

C'è una grande differenza tra una variabile semplice e una variabile oggetto, la variabile oggetto non è che un puntatore all'interno della memoria e si dovrà creare esplicitamente un oggetto e salvare la sua posizione nella variabile oggetto. Questo processo si chiama creare una nuova istanza di un oggetto o*istanziare un oggetto*.
Poiché gli oggetti sono diversi dalle variabili, Visual Basic for Applications utilizza una speciale istruzione chiamata istruzione *Set* che ha due forme.

1) *Set VariabileOggetto = New NomeClasse*

In questa forma, l'istruzione Set crea un nuovo oggetto basato su *NomeClasse*, ciò significa che Visual Basic allocherà memoria per l'oggetto e salverà la posizione in memoria nella classe *VariabileOggetto*.

2) *Set VariabileOggetto = EspressioneOggetto*

Nella seconda forma, l'istruzione *Set* fa due cose: prima di tutto rilascia l'oggetto a cui puntava, quindi salva un puntatore a un oggetto già esistente in *VariabileOggetto*.

La proprietà *Instancing* di un modulo di classe è impostata su *Private* di default che non permette a un progetto esterno di lavorare con le istanze di quella classe, si deve impostare la proprietà *Instancing* per *PublicNotCreatable* per consentire a progetti esterni, con un riferimento al progetto contenente la classe definita, per accedere e utilizzare istanze della classe. Notare che l'impostazione di PublicNotCreatable ancora non consente al progetto esterno di istanziare (cioè creare o chiama all'esistenza) l'oggetto classe o un'istanza della classe, che può essere istanziato solo dal progetto che contiene la definizione della classe. Si noti che il progetto esterno può utilizzare un'istanza della classe definita se il progetto di riferimento ha già creato tale istanza.

Come già accennato, in VBA è possibile creare i propri oggetti personalizzati attraverso la definizione di classi. Una classe viene creata inserendo un modulo di classe e per accedere alle proprietà e metodi della classe dell'oggetto da una routine in un modulo di codice, è necessario creare una nuova istanza dell'oggetto di classe. Si noti che possono essere create pluralità di istanze a un oggetto di classe. Ci sono due modi per creare un'istanza, uno con un codice a due-line o in alternativa con un codice a riga singola.

Per creare un'istanza di una classe Two-line si deve utilizzare l'istruzione Dim per creare una variabile (studente) e definirlo come un riferimento alla classe (Studenti): *Dim studente As Studenti*, mentre per creare un nuovo riferimento oggetto utilizzando la parola chiave *New*, indicando il nome della classe (Studenti) che si desidera creare un'istanza, dopo la parola chiave New: *Set studente = New Studenti*

In alternativa si può usare un codice a linea singola per creare un'istanza di una classe, in questo caso l'oggetto *Studenti* viene istanziato solo quando il metodo di classe viene chiamato *Dim studente As New Studenti*

Creare Proprietà della classe

Un modo per creare una proprietà di classe è quello di dichiarare una variabile pubblica nel modulo di classe, e questa proprietà sarà in lettura-scrittura, l'altro modo è quello di utilizzare procedure di proprietà per creare una variabile privata per contenere i valori e utilizzare istruzioni di proprietà (cioè Property Let, Property Set e Property Get). La creazione di proprietà mediante una variabile pubblica, anche se semplice, non è generalmente preferibile perché non è flessibile, mentre invece utilizzando le dichiarazioni di proprietà sarà possibile impostare una proprietà di sola lettura o sola scrittura in aggiunta a lettura-scrittura, mentre l'uso di una variabile pubblica creerà solo le proprietà in lettura-scrittura. Inoltre, utilizzando le istruzioni della struttura è possibile eseguire il codice per calcolare i valori come proprietà che utilizza, mentre una variabile pubblica non consente l'uso di codice per impostare o restituire il valore di una proprietà.

Creare metodi in un modulo di classe

Oltre alle proprietà, gli oggetti possono anche avere uno o più metodi, un metodo è definito come *Sub* e *Funzioni* ed è creato con le procedure di sub-routine e funzioni. Un metodo è una sub-routine contenente un insieme di codici che eseguono un'azione o un'operazione sui dati all'interno della classe, o una funzione che contiene un insieme di codici che restituisce un valore dopo l'esecuzione di un'operazione. In un modulo di classe, solo se il metodo è dichiarato pubblico può essere chiamato da un'istanza di questa classe, altrimenti se un metodo viene dichiarato privato può essere chiamato solo da altri metodi all'interno della classe. Si noti, che di default la procedura è pubblica se non vengono specificate le parole chiave pubbliche o private.

Utilizzo di procedure di proprietà

Le procedura di proprietà sono un insieme di codici che creano e manipolano le proprietà personalizzandole per un modulo di classe. Una procedura *Property*è dichiarata da una dichiarazione *Property Set*, *Property Let* o *Property Get* e termina con un'istruzione *End Property*. *Property Let* viene utilizzata per assegnare un valore di *sola scrittura* a una proprietà e *Property Get* restituisce o recupera il valore di una proprietà di *sola lettura*, che può essere solo restituito, ma non impostato. *Property Set* assegna un valore di *sola scrittura* e viene utilizzata per impostare un riferimento a un oggetto. Le procedure di proprietà sono solitamente definite in coppia, *Property Let* e *Property Get* o *Property Set* e *Property Get*, si tenga presente che una procedura creata con*Property Let* consente all'utente di modificare o impostare il valore di una proprietà, mentre l'utente non può impostare o modificare il valore di una proprietà di sola lettura (cioè Property Get).

Una procedura di proprietà può fare tutto ciò che è consentito all'interno di una routine, come eseguire un'azione o un calcolo sui dati. Una procedura Property Let (o Property Set) è una procedura indipendente che può passare argomenti, eseguire azioni come da un insieme di codici e cambiare il valore dei suoi argomenti, come una routine Get o una funzione, ma non restituisce un valore come loro. Una procedura per ottenere la proprietà è anche una procedura indipendente che possa passare argomenti, eseguire azioni come da un insieme di codici e cambiare il valore dei suoi argomenti, come una procedura Property Let (o Property Set), e può essere usata in modo simile a una funzione che ritorni il valore di una proprietà.

Una procedura Property Get accetta un argomento in meno della dichiarazione Property Let o Property Set e deve essere dello stesso tipo di dati come il tipo di dati dell'ultimo argomento nella dichiarazione Property Let o Property Set associata. La dichiarazione Property Get utilizzerà lo stesso nome di proprietà come utilizzato nella dichiarazione Property Let o Property Set associata.

Una procedura Property Let può accettare più argomenti, e in questo caso l'ultimo argomento contiene il valore da assegnare alla proprietà, questo ultimo argomento nella lista degli argomenti è il valore della proprietà impostata dalla procedura chiamante. Il nome e il tipo di dati di ogni argomento in una procedura Property Let e il suo corrispondente nella routine Property Get dovrebbe essere la stessa, fatta eccezione per l'ultimo argomento nella procedura Property Let che è supplementare. Nel caso di una procedura Property Let con un singolo argomento (è richiesto almeno un argomento da definire), questo argomento contiene il valore da assegnare alla proprietà ed è il valore fissato dalla procedura chiamante, in questo caso la procedura Property Get non avrà alcun argomento.

Una procedura Property Set può accettare più argomenti, e in questo caso l'ultimo argomento contiene il riferimento all'oggetto effettivo per la proprietà. Nel caso di una procedura Property Set con un singolo argomento (è richiesto almeno un argomento da definire), questo argomento contiene il riferimento all'oggetto per la proprietà. Il tipo di dati dell'ultimo argomento o il singolo argomento deve essere un tipo di oggetto o un valore Variant.

La procedura Property Set è simile e una variazione della procedura Property Let ed entrambi vengono utilizzati per impostare i valori. Una procedura Property Set viene utilizzata per creare le proprietà degli oggetti che sono in realtà puntatori ad altri oggetti, mentre una procedura Property Let assegna i valori alle proprietà scalari come stringhe, interi, date, etc.

Di seguito è riportata la sintassi per le dichiarazioni delle tre Procedura di proprietà.

Property Get
Property Get PropertyName(argomento_1, argomento_2, …, argomento_n) As Type

Property Let
Property Let PropertyName(argomento_1, argomento_2, …, argomento_n+1)

Property Set
Property Set PropertyName(argomento_1, argomento_2, …, argomento_n+1)

Esempio: Creare una classe Properties utilizzando procedure di proprietà. *Inserire il codice in un modulo di classe e denominarlo class_Stud*

```
Private nomeStud As String, votoStud As Double
Public Property Let Name(strN As String)
'si dichiara public in modo che possa essere chiamata da un'istanza in un altro modulo
   nomeStud = strN
End Property

Public Property Get Name() As String
'restituisce la proprietà Name
   Name = nomeStud
End Property

Public Property Let voto(ivoto As Double)
'assegna la proprietà voti
   votoStud = (ivoto / 80) * 100
End Property

Public Property Get voto() As Double
'restituisce la proprietà voti
   voto = votoStud
End Property

Public Function valuT() As String
'si dichiara public in modo che possa essere chiamata da un'istanza in un altro modulo.
   Dim valuta1 As String

   If votoStud >= 80 Then
      valuta1 = "A"
   ElseIf votoStud >= 60 Then
      valuta1 = "B"
   ElseIf votoStud >= 40 Then
      valuta1 = "C"
   Else
      valuta1 = "Bocciato"
   End If

valuT = valuta1
End Function
```

Inserire il codice sotto riportato in un modulo standard

```
Sub studenti()
'si utilizza l'istruzione Dim per creare una variabile e definirla come un riferimento alla classe.
    Dim studente As class_Stud
'si crea un nuovo riferimento a un oggetto utilizzando la parola chiave New. Indicare il nome
della classe a cui si desidera creare un'istanza, dopo la parola chiave New
Set studente = New class_Stud
    studente.Name = "Marco"
'si richiama la proprietà Name nell'oggetto studente
    MsgBox studente.Name
'si imposta la proprietà voto nell'oggetto studente al valore 45, e passa questi dati alla variabile
` i voti nella proprietà voto
studente.voto = 45
MsgBox studente.voto
MsgBox studente.valuT
MsgBox studente.Name & " ha ottenuto il " & studente.voto & " % di voti con valutazione " &
studente.valuT
End Sub
```

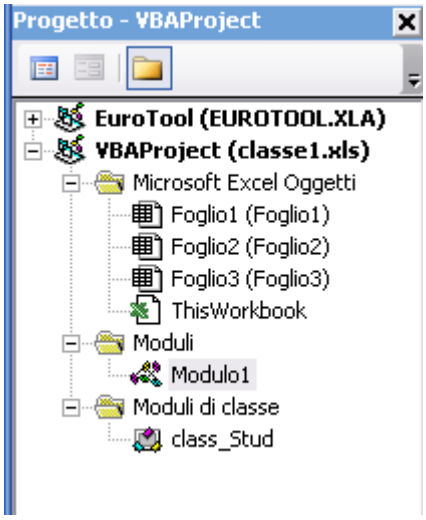


Fig. 1

Esempio: Creare una classe utilizzando la procedura Property Let passando più argomenti. *Inserire il codice in un modulo di classe e denominarlo class_rettangolo*

Nota: *Una procedura Property Let può accettare più argomenti, e in questo caso l'ultimo argomento contiene il valore da assegnare alla proprietà. Questo ultimo argomento nella lista degli argomenti è il valore della proprietà impostata dalla procedura chiamante. Il nome e il tipo di dati di ogni argomento in una procedura Property Let e il suo corrispondente la routine Property Get dovrebbe essere la stessa, fatta eccezione per l'ultimo argomento nella procedura Property Let che è opzionale.*

```
Private Are_a As Double
Public Property Let Area(Ba1 As Double, Alt1 As Double, ar As Double)
Are_a = ar
MsgBox "Argomenti ricevuti - Ba1: " & Ba1 & ", Alt1: " & Alt1 & ", ar: " & ar
End Property
Public Property Get Area(Ba1 As Double, Alt1 As Double) As Double
Area = Are_a
End Property
```

Inserire il codice sotto riportato in un modulo standard

```
Sub rettangolo()
Dim Ba As Double, Alt As Double
Dim rett As New class_rettangolo
Ba = InputBox("Inserire Base del Rettangolo")
Alt = InputBox("Inserire Altezza del Rettangolo")
rett.Area(Ba, Alt) = Ba * Alt
a = rett.Area(Ba, Alt)
MsgBox "L'area del Rettangolo con Base " & Ba & ", e Altezza " & Alt & ", è " & a
End Sub
```

Esempio: Creazione di sola lettura Classe proprietà con solo il blocco PropertyGet_EndProperty. *Inserire il codice in un modulo di classe e denominarlo classRettangolo*

```
Private rett_B As Double, rett_H As Double
Public Property Let Base(k As Double)
   rett_B = k
End Property

Public Property Get Base() As Double
   Base = rett_B
End Property

Public Property Let Altez(k1 As Double)
   rett_H = k1
End Property

Public Property Get Altez() As Double
   Altez = rett_H
End Property

Public Property Get area_r() As Double
   area_r = Base * Altez
End Property
```

Inserire il codice sotto riportato in un modulo standard

```
Sub rettangolo1()
Dim a As Double, b As Double
Dim areaRett As New classRettangolo
a = InputBox("Inserire Base rettangolo")
b = InputBox("Inserire Altezza rettangolo")

areaRett.Base = a
areaRett.Altez = b
MsgBox areaRett.area_r
End Sub
```

Esempio: Utilizzare Property Set per impostare un riferimento a un oggetto. *Inserire il codice in un modulo di classe e denominarlo classeAuto*

```
Private tipoVet As classeTipoAut
Public Property Set tipo(objtipo As classeTipoAut)
   Set tipoVet = objtipo
End Property

Public Property Get tipo() As classeTipoAut
   Set tipo = tipoVet
End Property
```

Inserire il codice sotto riportato in un modulo di classe e denominarlo classeTipoAut

```
Private tipoCol As String, tipoNom As String, KL As Double
Property Let colore(col As String)
   tipoCol = col
End Property
Property Get colore() As String
   colore = tipoCol
End Property
Property Let nome(nom As String)
   tipoNom = nom
End Property
Property Get nome() As String
   nome = tipoNom
End Property
Property Let percorso(kmLit As Double)
   KL = kmLit
End Property
Property Get percorso() As Double
   percorso = KL
End Property
Function costoBenz(cost_benz As Double, distanza As Double) As Double
   costoBenz = (distanza / percorso) * cost_benz
End Function
```

Inserire il codice sotto riportato in un modulo standard

```
Sub propSetCars()
Dim dist As Double, cost As Double
Dim auto As classeAuto
Set auto = New classeAuto
Set auto.tipo = New classeTipoAut
auto.tipo.colore = "Nero"
auto.tipo.nome = "Toyota Yaris"
auto.tipo.percorso = 50
dist = InputBox("Inserisci i chilometri percorsi in un mese dalla vettura")
cost = InputBox("Inserisci il costo del carburante al litro")
MsgBox "Il colore della vettura è : " & auto.tipo.colore
MsgBox "Il modello della vettura è : " & auto.tipo.nome
MsgBox "La tua auto percorre " & auto.tipo.percorso & " Km. con 1 litro di carburante"
MsgBox auto.tipo.costoBenz(dist, cost) & " Eur." & " è il costo mensile del carburante"
End Sub
```

Esempio: Utilizzo dell'istruzione Property Set per impostare un riferimento a un oggetto Range. Inserire il codice nel modulo di classe denominato classeRange:

```
Private coloreSF As Integer, SNome As String, vRng As Range
Public Property Set Range_At(oRng As Range)
   Set vRng = oRng
End Property

Public Property Get Range_At() As Range
   Set Range_At = vRng
End Property

Property Let Nome(nom As String)
   SNome = nom
End Property

Property Get Nome() As String
   Nome = SNome
End Property

Property Let colore(col As Integer)
   coloreSF = col
End Property

Property Get colore() As Integer
   colore = coloreSF
End Property

Sub Mcolore()
   Range_At.Interior.ColorIndex = colore
End Sub
```

Inserire il codice sotto riportato in un modulo standard

```
Sub classe_Range()
Dim RangeATT As classeRange
Set RangeATT = New classeRange
Set RangeATT.Range_At = ActiveCell
RangeATT.colore = 5

If RangeATT.colore < 1 Or RangeATT.colore > 56 Then
MsgBox "Errore! Inserisci un valore per il colore compreso tra 1 e 56"
Exit Sub
End If

RangeATT.Mcolore
MsgBox "colore di sfondo: ColorIndex " & RangeATT.colore & "  Inserito nella cella " &
RangeATT.Range_At.Address

End Sub
```

Eventi di Classe Personalizzati

Un codice VBA viene attivato quando si verifica un evento come, cliccare su un pulsante, aprire la cartella di lavoro, selezionare una cella o cambiare una selezione di celle in un foglio di lavoro, e così via. Excel ha anche le sue procedure di eventi che vengono attivati da un evento predefinito e vengono installati all'interno di Excel con un nome standard e predeterminato, come la procedura di modifica del foglio di lavoro che viene installato con il foglio di lavoro come "Private Sub Worksheet_Change (ByVal Target As Range)". Quando il contenuto di una cella del foglio di lavoro cambia, VBA chiama la routine evento Worksheet_Change ed esegue il codice che contiene. Ora vediamo come è possibile creare i propri eventi personalizzati in una classe.

Definire un evento personalizzato

Il primo passo è quello di dichiarare l'evento nella sezione di dichiarazione della classe usando la parola chiave **Event** per definire un evento personalizzato in un modulo di classe. Questa dichiarazione può avere qualsiasi numero di argomenti, e deve essere dichiarata come *Public* per renderlo visibile all'esterno del modulo oggetto. Si noti che è possibile dichiarare e generare eventi solo nei moduli oggetto ThisWorkbook, fogli di lavoro e fogli grafici, moduli Form e moduli di classe, e non da un modulo di codice standard.

Generare un evento

Dopo aver dichiarato un evento, si deve utilizzare una dichiarazione **RaiseEvent** per attivare l'evento dichiarato e la procedura di evento viene eseguita quando viene generato o attivato l'evento. Si ricorda che l'evento è dichiarato in un procedimento pubblico all'interno del modulo di classe utilizzando la parola chiave Evente con la dichiarazione *RaiseEvent* vengono passati i valori agli argomenti alla routine evento che viene eseguita

Codice esterno per generare l'evento

Abbiamo bisogno di un codice esterno per chiamare la procedura pubblica nel modulo di classe, che genera l'evento e tramite questo codice si determina quando l'evento verrà generato mediante il quale la procedura di evento viene eseguita.

Creare una routine evento

Si usa la parola chiave **WithEvents** per dichiarare una variabile oggetto della classe personalizzata in cui è definito l'evento personalizzato, dichiarando questa variabile oggetto, l'istanza della classe personalizzata punta alle variabili che risponderanno all'evento aggiungendo l'oggetto alla lista degli eventi nella finestra del codice. Solo le variabili dichiarate a livello di modulo possono essere utilizzato con la parola chiave WithEvents. Inoltre, le variabili possono essere dichiarate utilizzando le parole chiave WithEvents solo in moduli di oggetti e non in un modulo di codice standard. Dopo la dichiarazione della variabile oggetto, la procedura di evento può essere creata in modo simile alle procedure standard di VBA - la variabile oggetto verrà visualizzato nell'elenco a discesa Oggetti e tutti i suoi eventi sono elencati nell'elenco a discesa della procedura.

Esempio: Creare un evento personalizzato utilizzando una procedura Worksheet_Change per attivare l'evento personalizzato. Inserire il codice sotto riportato nel modulo di classe denominato classe_Range:

```
Private RngV As Range, coloreS As Integer, nomeS As String
Public Event selezionaC(cell As Range)

Public Property Set RangeS(objRng As Range)
   Set RngV = objRng
RaiseEvent selezionaC(RngV)
End Property

Public Property Get RangeS() As Range
Set RangeS = RngV
End Property

Property Let nome(nom As String)
nomeS = nom
End Property

Property Get nome() As String
nome = nomeS
End Property

Property Let colore(col As Integer)
coloreS = col
End Property

Property Get colore() As Integer
colore = coloreS
End Property

Sub AssColore()
RangeS.Interior.ColorIndex = colore
End Sub
```

Inserire il codice sotto riportato nel modulo del Foglio 1

```
Private WithEvents rang As classe_Range
Private Sub rang_selezionaC(cell As Range)
    rang.colore = 4
If rang.colore < 1 Or rang.colore > 56 Then
MsgBox "Errore! Inserire un valore valido per ColorIndex compreso tra 1 e 56"
Exit Sub
End If
rang.nome = "Prima Cella"

rang.AssColore
Dim i As Integer
i = rang.colore
rang.RangeS.Select
Selection.Offset(0, 1).Value = "Nome : " & rang.nome
Selection.Offset(0, 2).Value = "Indirizzo : " & Selection.Address
Selection.Offset(0, 3).Value = "Colore di sfondo : " & i
Selection.Offset(0, 4).Value = "Contenuto : " & Selection.Value
End Sub

Private Sub Worksheet_Change(ByVal Target As Range)
    On Error GoTo ErrorHandler
Set rang = New classe_Range
If Target.Address = Range("A1").Address Then
Set rang.RangeS = Target
Else
Exit Sub
End If

ErrorHandler:
  Application.EnableEvents = True

End Sub
```

Esempio: Creare un evento personalizzato: All'inizializzazione della Form si attiva l'evento personalizzato.

```
Private tb As MSForms.TextBox, strSeq As String
Public Event eTxtBx(objTxtBx As MSForms.TextBox)
Public Property Set Stesto(objTxtBx As MSForms.TextBox)
    Set tb = objTxtBx
RaiseEvent eTxtBx(tb)
End Property
Public Property Get Stesto() As MSForms.TextBox
    Set Stesto = tb
End Property
Property Let sequenza(tbSeq As String)
    strSeq = tbSeq
End Property
Property Get sequenza() As String
    sequenza = strSeq
End Property
```

Inserire una Form con 2 textBox (TextBox1 e TextBox2) e un CommandButton (CommandButton1) all'interno del modulo, inserire il seguente codice

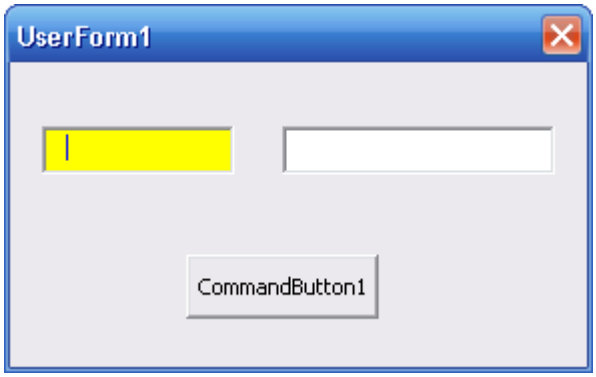


Fig. 2

```
Private WithEvents tx As classeTextBox, sq1 As String, sq2 As String
Private Sub CommandButton1_Click()
Dim objControl As Control
For Each objControl In Me.Controls
If TypeName(objControl) = "TextBox" Then
    If Not objControl.Name = "TextBox1" Then
    objControl.Value = "Valore Copiato : " & tx.Stesto.Value
    objControl.BackColor = vbRed
    End If
End If
Next
MsgBox "Testo Copiato dal " & sq1 & " Al " & sq2
End Sub
Private Sub TextBox1_Change()
    If tx.Stesto.Value = "" Then
        tx.Stesto.BackColor = vbYellow
    Else
        tx.Stesto.BackColor = vbGreen
    End If
End Sub
Private Sub tx_eTxtBx(objTxtBx As MSForms.TextBox)
tx.Stesto.BackColor = vbYellow
With Me.TextBox1
tx.sequenza = "Primo TextBox"
sq1 = tx.sequenza
End With
With Me.TextBox2
tx.sequenza = "Secondo TextBox"
sq2 = tx.sequenza
End With
End Sub
Private Sub UserForm_Initialize()
Set tx = New classeTextBox
Set tx.Stesto = Me.TextBox1
End Sub
```

Esempio: Creare un evento personalizzato: - utilizzare la parola chiave WithEvents, all'interno del modulo di classe, per dichiarare una variabile oggetto. Inserire il codice nel modulo di classe denominato classeCombo

```
Public WithEvents ComboB As MSForms.ComboBox
Public Sub SCombo(objCbx As MSForms.ComboBox)
    Set ComboB = objCbx
End Sub
Public Sub ComboB_AddItem(strItem As String, Cancel As Boolean)
    If Cancel = False Then
        ComboB.AddItem strItem
    End If
If strItem <> "" Then
    ComboB.BackColor = vbGreen
End If
End Sub
Private Sub ComboB_Change()
    If ComboB.Value = "" Then
        ComboB.BackColor = vbWhite
    End If
End Sub
```

Inserire una Form, con un ComboBox (ComboBox1) e un CommandButton (CommandButton1) all'interno del modulo. Inserire codice sotto riportato nel modulo della Form

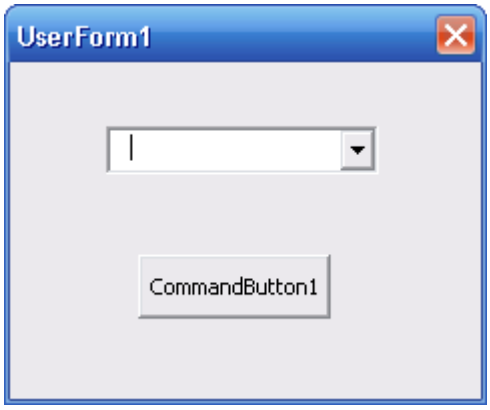


Fig. 3

```
Private cB As New classeCombo
Private Sub CommandButton1_Click()
Dim Stesto As String
    Stesto = cB.ComboB.Text
    Dim Cancel As Boolean
mess = MsgBox("Confermi di voler aggiungere la voce digitata al ComboBox?", vbYesNo)
If mess = vbNo Then
    Cancel = True
End If
Call cB.ComboB_AddItem(Stesto, Cancel)
End Sub
Private Sub UserForm_Initialize()
    cB.SCombo Me.ComboBox1
End Sub
```

Creare e richiamare procedure Sub e Funzioni

Se volete sviluppare una applicazione VBE, una delle prime cose che dovete sapere è la differenza tra *Funzioni* e *Subroutine*, quest'ultime note anche come *Procedure Sub*. In entrambi i casi si tratta di una sorta di "raggruppamento" di istruzioni che svolgono un'operazione comune, e per questo sono molto simili, praticamente tutto il codice di un programma è contenuto all'interno di funzioni e routine. Conoscere la differenza tra i due tipi di procedure aiuterà a prendere la decisione giusta su quale usare, in quanto consentono di suddividere il codice in blocchi, ognuno dotato di una propria specifica funzionalità, evitando ripetizioni e garantendo una miglior leggibilità del codice stesso. La differenza fondamentale tra le due è che le procedure eseguono operazioni di tipo generale, come una serie di istruzioni collegate ad un evento o richiamate nel codice principale, mentre le funzioni rappresentano un procedimento di calcolo o una elaborazione che deve restituire un valore come risultato al programma chiamante.

Se si scrive lo stesso codice più volte, l'applicazione che state costruendo potrebbe beneficiare di una procedura, piuttosto che duplicare il codice in più posizioni, che oltre a rendere più difficoltosa l'operazione di *Debug* rende il programma stesso più grande di quanto dovrebbe essere. Una procedura è indicata anche con il termine di *macro*, che è rappresentata come un insieme di codici che rendono Excel in grado di eseguire un'azione. Una procedura è una sequenza di istruzioni che inizia con la parole chiave *Sub*, seguita dal nome della routine, e poi da due parentesi, al cui interno è possibile inserire dei parametri (opzionali) richiesti della procedura, ai quali non c'è limite, come numero, e termina con la parola chiave *End Sub*

La procedura Sub

Sub è la forma breve di sotto-routine, ed è utilizzata per gestire una certa attività all'interno di un programma. Le subroutine vengono usate per dividere i task in singole procedure, cosa molto utile in quanto la divisione di un programma in procedure e sotto procedure lo rende più leggibile e riduce le probabilità di errore. Le subroutine possono accettare alcuni argomenti come parametri ma non restituiscono nessun valore alla procedura o alla funzione chiamante. Una procedura Sub inizia con la parola chiave *Sub*, seguita dal nome e poi da una serie di parentesi e termina con l'istruzione *End Sub* nel mezzo tra le due parole chiave va inserito il codice. La sintassi è:

```
Sub nome_routine ()
'istruzioni della subroutine
End Sub
```

Oppure, nel caso si voglia passare dei parametri alla Sub si usa questa sintassi:

```
Sub nome_routine (param1 As tipo, param2 As tipo)
 'istruzioni della subroutine
End Sub
```

Dove *param1* e *param2* rappresentano i parametri che vengono passati alla procedura. Si noti come la dichiarazione di una subroutine è molto simile a come si dovrebbe dichiarare una variabile, specificando il nome del parametro e il tipo di dati, dove ciascun parametro passato alla procedura è trattato come una variabile locale nella subroutine, il che significa che la "durata" del parametro è la stessa di quella della procedura. Ci sono due modi per passare una variabile in ingresso ad una procedura: per *valore* (o ByVal) oppure per *riferimento* (o ByRef). La differenza fra le due modalità sta nel rendere disponibile alla procedura o una copia del valore della variabile, o il contenuto vero e proprio della stessa. Sintatticamente si indica la modalità per valore o per riferimento premettendo *ByVal* o *ByRef* rispettivamente prima della dichiarazione dei parametri di ingresso

Sub nome_routine (ByVal As Type)
Sub nome_routine (ByRef As Type)

Si consideri che in assenza del tipo di dati nella dichiarazione della routine, per impostazione predefinita, una subroutine tratta gli argomenti passati per valore, il che significa che la procedura non può modificare il contenuto della componente variabile nel codice chiamante. Per richiamare una subroutine che non prevede il passaggio di parametri da altra routine, si può alternativamente utilizzare l'istruzione ***Call*** seguita dal nome della routine oppure indicare il solo nome della routine da eseguire.

```
Call aggiungi1
aggiungi1
```

Nel caso si debba richiamare una routine che necessita il passaggio di parametri allora è necessario ricorrere all'istruzione *Call* e inserire, dopo il suo nome, i valori separati da una virgola e racchiusi tra parentesi.

Call aggiungi1(x)

Si consideri il seguente programma come esempio:

```
Sub prova1()
Dim x As Integer
x = 5
MsgBox x
Call aggiungi1(x)
MsgBox x
End Sub

Sub aggiungi1(ByVal i As Integer)
i = i + 1
End Sub
```

In questo esempio viene assegnato il valore 5 alla variabile **x**, e poi tramite la funzione *MsgBox* ne viene stampato il valore a video, successivamente viene chiamata la procedura *aggiungi1* che prende in ingresso una copia del valore di **x** e lo incrementa. Questa operazione però non ha effetto sul contenuto della variabile x infatti quando successivamente viene stampato il valore di x rimane sempre 5. Se invece consideriamo il seguente listato

```
Sub prova2()
Dim x As Integer
x = 5
MsgBox x
Call aggiungi2(x)
MsgBox x
End Sub

Sub aggiungi2(ByRef i As Integer)
i = i + 1
End Sub
```

Questa volta la variabile **x** è stata passata per riferimento (ByRef), questo significa che la procedura va a modificare il contenuto della variabile, Infatti alla fine dell'elaborazione il valore di x è stato incrementato e vale 6

La procedura Function

Abbiamo già detto che una procedura sub è un insieme di codici VBA o istruzioni che svolgono un'azione, ma non *restituiscono un valore*, mentre invece una funzione è anche usata per eseguire un'azione o un calcolo, cosa che può essere eseguita anche da una sub routine, ma la Function *restituisce un valore*. Il nome della funzione può essere utilizzato nell'ambito della procedura per indicare il valore restituito dalla funzione stessa e non può essere creata utilizzando il registratore di Macro. Una routine *Function* può essere richiamata utilizzando il nome in un'espressione che non è possibile per una procedura sub, che invece necessita di essere richiamata solo da una dichiarazione autonoma. Poiché una funzione restituisce un valore, può essere utilizzata direttamente in un foglio di lavoro inserendo il nome della funzione dopo aver digitato un segno di uguale

Una Function inizia con la parola chiave *Function*, seguito da un nome e poi da una serie di parentesi, inoltre è necessario specificare il tipo di dati che corrisponde al tipo di valore che la funzione restituirà, digitando la parola chiave **As** dopo le parentesi seguita dal tipo di dati.

```
Function nome_funzione(param1 As tipo_param1, param2 As tipo_param2)  As tipo_valore
... istruzioni funzione
End Function
```

Esempio di funzione:

```
Function calcola_area(Raggio As Double) As Double
 calcola_area = Raggio * Raggio * 3.14
End Function
```

Le funzioni possono essere create senza argomenti e con qualsiasi numero di argomenti che sono elencati tra le parentesi e in caso di argomenti multipli, si deve separarli con una virgola e termina con un'istruzione ***End Function*** che può essere digitato, andando alla riga successiva dopo aver digitato la dichiarazione viene inserita automaticamente da VBE. Le istruzioni espresse in codice VBA vanno collocate tra la dichiarazione *Function* e la dichiarazione finale *End Function*. La funzione può essere richiamata da un'altra procedura, vale a dire una sub o una function e può essere utilizzata direttamente in una formula del foglio di lavoro inserendo il nome della funzione dopo aver digitato un segno di uguale. Vediamo alcuni esempi di Funtion e come si chiamano:

Esempio: Utilizzare una function per inserire il valore restituito in una cella del foglio attivo

```
Function part_nome() As String
Dim nome As String
Dim cognome As String
nome = InputBox("Inserire nome")
cognome = InputBox("Inserire il cognome")
part_nome = nome & "" & cognome
End Function

Sub nome_int()
ActiveSheet.Range("A1") = part_nome
End Sub
```

Esempio: Usare una Function per fare calcoli e ricevere il risultato come valore ritorno

```
Function area1() As Double
Dim i As Integer
i = InputBox("Immettere il raggio del cerchio")
area1 = i * i * 3.14
area1 = Format(area1, "#. # #")
End Function

Sub calcola_1()
MsgBox "L'area del cerchio è:" & " " & area1
End Sub
```

Esempio: Passare una variabile alla Function per eseguire dei calcoli

```
Function tre(i As Integer) As Long
tre = i * 3
End Function

Sub chiama_3()
Dim a As Integer
a = InputBox("Immettere un numero intero")
MsgBox tre(a)
End Sub
```

Si può osservare che quando la Function non riceve argomenti l'insieme di parentesi, dopo il nome della procedura, è vuoto, tuttavia, quando gli argomenti sono passati a una procedura secondaria o a una funzione da altre procedure, allora questi sono elencati tra le parentesi.

Esempio: La funzione calcola_1 restituisce un valore che può essere utilizzato in una sub

```
Function calcola_1() As Double
Dim a As Double
Dim b As Double
a = InputBox("Inserire il primo numero")
b = InputBox("Inserire il secondo numero")
calcola_1 = (a + b) * 2
End Function

Sub calcola()
Dim c As Double
c = calcola_1 * 5 / 2
MsgBox c
End Sub
```

Esempio: La funzione calcola_2 non restituisce un valore che può essere utilizzato dalla procedura chiamante.

```
Sub calcola_2()
Dim a As Double
Dim b As Double
Dim c As Double
a = InputBox("Inserire il primo numero")
b = InputBox("Inserire il secondo numero")
c = (a + b) * 2
MsgBox c
End Sub

Sub usa_calcola2()
calcola_2
End Sub
```

Regole e Convenzioni per assegnare il nome alle procedure

Ci sono alcune regole che devono essere seguite quando si assegna il nome alle procedure, una di queste è quella di assegnare un nome che rifletta l'azione che eseguirà la stessa, si può usare una frase differenziando le parole da un carattere di sottolineatura o una lettera maiuscola: esempio *cerca_N* può equivalere a cerca nomi, si sconsiglia di usare nomi molto lunghi, inoltre il nome deve iniziare con una lettera, sono da evitare i numeri o un carattere di sottolineatura. Un nome può essere costituito da lettere, numeri o caratteri di sottolineatura, ma non può avere un periodo o caratteri di punteggiatura o caratteri speciali: esempio (.) @ # $% ^ & * () + - = [] {}; ':! ",. / <> \ |? `~ .

Il nome dovrebbe essere costituito da una stringa di caratteri continui, senza spazio intermedio e può avere un massimo di 255 caratteri, inoltre i nomi delle procedure non possono utilizzare parole chiave o riservate come And, Or, Loop, Do, Len, Close, data, ElseIf, Else, Select, etc. che VBA utilizza come parte del suo linguaggio di programmazione.

Procedure Public o Private

Una procedura VBA può essere utilizzata in ambito pubblico o privato e questo metodo di procedura può essere specificato con la parola chiave *Public* o *Private*, se non viene specificato il metodo, VBA tratta le procedure come pubbliche di default. Una procedura dichiarata come Private può essere richiamata solo da tutte le procedure nello stesso modulo e non saranno visibili o accessibili alle procedure di moduli esterni nel progetto e non apparirà nella finestra di dialogo Macro. La Procedura pubblica invece può essere richiamata da tutte le procedure dello stesso modulo, ma anche da tutte le procedure di moduli esterni nel progetto e il suo nome verrà visualizzato nella finestra di dialogo Macro e può essere eseguito da esso.

Esempi di una procedura pubblica:
Sub calcola1 ()
Public Sub calcola1 ()
Function calcola1 () As Double
Public Function calcola1 () As Double

Esempi di una procedura privata:
Private Sub calcola1 ()
Private Function calcola1 () As Double

Moduli Standard

Questi sono anche indicati come moduli di codice o semplicemente moduli, e ciascun modulo può essere utilizzato per coprire un certo aspetto del progetto. La maggior parte del codice VBA e le funzioni personalizzate (cioè Funzioni definite dall'utente denominate UDF) sono collocati in moduli di codice standard che non vengono utilizzati per le procedure di evento o per gli eventi di applicazioni create in un modulo di classe dedicato. Per quanto riguarda gli eventi non connessi con oggetti, come il metodo *OnTime* (attiva automaticamente un codice VBA ad intervalli periodici o in un giorno o momento specifico) e il metodo *OnKey* (eseguire un codice specifico su pressione di un tasto o una combinazione di tasti), essendo questi metodi non associati a un particolare oggetto il loro codice viene inserito in un modulo standard.

Modulo Workbook

ThisWorkbook è il nome del modulo per la cartella di lavoro e viene utilizzato per inserire eventi nella cartella di lavoro e eventi Application. Gli eventi *Workbook* sono azioni connesse con la cartella di lavoro per innescare un codice VBA o macro, vale a dire, apertura, chiusura, salvataggio, attivazione e disattivazione del foglio di lavoro sono esempi di eventi delle cartelle di lavoro. Con l'evento *Open* della cartella di lavoro, è possibile eseguire una sub automaticamente quando una cartella di lavoro è aperta e il codice dell'evento della cartella di lavoro deve essere inserito nel modulo di codice per l'oggetto ThisWorkbook, mentre se sono posti in moduli di codice standard Excel non sarà in grado di trovarli ed eseguirli. Anche se eventi Application possono essere creati in qualsiasi modulo oggetto, è meglio che siano posizionati in un modulo oggetto ThisWorkbook oppure è possibile creare un modulo di classe per gestirli.

Modulo Foglio

Un modulo foglio ha lo stesso nome del foglio di lavoro con cui è associato (Foglio1, Foglio2 etc.) e in VBE, il codice del nome può essere modificato solo nella finestra *Proprietà* e non a livello di programmazione, inoltre il modulo foglio può essere per un foglio di lavoro o un grafico e viene usato per posizionare gli eventi innescati o da eventi associati al foglio di lavoro, o da macro. Per utilizzare una routine evento del foglio di lavoro, in VBE si deve selezionare il foglio di lavoro dalla casella *Progetti* e quindi selezionare una procedura corrispondente dalla finestra del *Codice* e una volta selezionato l'evento specifico, si deve inserire il codice VBA che si desidera eseguire. Si ricorda che se il codice viene immesso in un modulo standard, Excel non sarà in grado di trovare le istruzioni ed eseguirle. Le principali istanze di evento sono: la selezione di una cella o cambiando la selezione di celle in un foglio di lavoro, cambiare il contenuto di una cella, selezionare o attivare un foglio, calcolare un foglio di lavoro e così via. Ad esempio, con l'evento *Change*(modifica) di *Worskheet*, una procedura viene eseguita automaticamente quando si modifica il contenuto di una cella del foglio di lavoro.

Modulo UserForm

Procedure di eventi per *Userform*, o per i suoi controlli, sono posizionati nel modulo di codice della Form prescelta, e sono moduli pre-determinati, cioè che si verificano per un particolare uso della Form e/o dei relativi controlli, esempi in questo caso includono gli eventi: Initialize, Activate o Click. Per raggiungere un evento è necessario fare doppio clic sul corpo della Form per visualizzare il modulo di codice, quindi selezionare Userform o il suo controllo dalla casella *Oggetti* e quindi selezionare una procedura corrispondente dalla casella *Routine*. Dopo aver selezionato l'evento specifico, inserire il codice VBA che si desidera eseguire, ricordate di impostare la proprietà *Name* dei controlli prima di usare le routine di evento per loro, altrimenti sarà necessario cambiare il nome della procedura che corrisponde al nome del controllo.

Esecuzione di Function

La Function può essere chiamata da un'altra procedura vale a dire una procedura Sub o Function, e la funzione può essere utilizzata direttamente nel foglio come una formula inserendo il nome della funzione dopo aver digitato un segno di uguale.

Esecuzione di procedure di evento

Gli eventi sono azioni eseguite, o eventi, che innescano una macro VBA e vengono attivati quando si verifica un evento come apertura, chiusura, attivazione, disattivazione della cartella di lavoro, di una selezione di celle o di una sola cella, in pratica consiste nel fare un cambiamento del contenuto di un foglio di lavoro o di una cella. Le procedure di evento sono collegate agli oggetti, una procedura evento è una procedura con un nome standard che viene eseguito sul verificarsi di un evento corrispondente. Le procedure di evento sono attivate da un evento predefinito e vengono installate all'interno di Excel con un nome standard. Si consideri la procedura di modifica del foglio di lavoro che viene installata con il foglio di lavoro nella procedura *Private Sub Worksheet_Change (ByVal Target As Range)* che viene richiamata automaticamente quando un oggetto riconosce il verificarsi di un evento che possa modificare l'oggetto stesso. Una procedura evento per un oggetto è una combinazione del nome dell'oggetto (come specificato nella proprietà Name), un carattere di sottolineatura e il nome dell'evento.

Esecuzione routine Sub

Nella scheda *Visualizza* della barra multifunzione, cliccare su *Macro* nel gruppo Macro e poi su *Visualizza macro* che aprirà la finestra di dialogo macro. Nella finestra di dialogo Macro, selezionare il nome della macro e fare clic su *Esegui* per eseguire la Sub/macro. È inoltre possibile aprire la finestra di dialogo macro facendo clic su Macro nel gruppo Codice nella scheda Sviluppo sulla barra multifunzione oppure utilizzare la combinazione di tasti *Alt + F8* per aprire la finestra di dialogo macro.

Tasto di scelta rapida: Per utilizzare il tasto di scelta rapida associato alla macro è necessario avere già assegnato un tasto alla macro che può essere fatto sia nel momento in cui si inizia la registrazione di una macro o poi selezionando il nome della macro nell'elenco delle macro presenti nella finestra di dialogo Macro e facendo clic sul pulsante Opzioni .

Per eseguire una macro dall'editor di VBE si deve cliccare all'interno della procedura e premere F5 (o fare clic su Esegui nella barra dei menu) oppure dal menu Strumenti nella barra dei menu e cliccare su macro che apre la finestra di dialogo Macro, e selezionare il nome della macro e fare clic su Esegui per eseguirla. Ovviamente questi metodi non sono molto user friendly per eseguire le macro, un modo migliore per eseguire una macro potrebbe essere quella di fare clic su un pulsante accompagnate da un testo auto-esplicativo che appare sul foglio di lavoro

Si consideri che è possibile assegnare una macro a qualsiasi controllo modulo agendo dalla scheda Sviluppo sulla barra multifunzione, fare clic su Inserisci nel gruppo Controlli, selezionare e fare clic su pulsante nei controlli modulo e quindi fare clic sul foglio di lavoro in cui si desidera posizionare l'oggetto, poi di deve fare clic col destro del mouse sul pulsante e selezionare Assegna macro nella finestra che appare dalla quale è possibile selezionare e assegnare una macro al pulsante

Assegnare più Macro

È possibile assegnare più macro a un oggetto, un'immagine o un controllo, chiamando altre procedure secondarie. Consultare di seguito, ad esempio, nella procedura secondaria (CommandButton1_Click) che viene eseguito facendo clic su un pulsante di comando, altre due procedure secondarie denominate Macro1 e Macro2 sono chiamate ed eseguite. In questo esempio le macro sono chiamati con i loro nomi, ma è possibile visualizzare tutte le macro facendo clic su macro nel gruppo di macro (nella scheda Visualizza della barra multifunzione) e poi selezionando Visualizza macro. Ricordarsi di digitare i nomi di macro su righe separate durante la loro chiamata.

```
Private Sub CommandButton1_Click ()
Macro1
Macro2
End Sub
```

Editor di Visual Basic for Applications

Abbiamo visto nella lezione precedente come registrare ed eseguire una macro, ora vediamo l'Editor di Visual Basic che offre numerosi strumenti avanzati di sviluppo e programmazione per modificare o scrivere delle macro, creare finestre di dialogo personalizzate e tanto altro, consentendo di far interagire le applicazioni di Microsoft Office con moduli programmati. Faremo una panoramica generale dell'editor di VBA e approfondiremo i comandi dei suoi menù, dei pulsanti delle barre strumenti e di come essi si ambientano in Excel. Le macro in *Visual Basic for Applications* sono memorizzate in una speciale parte della cartella di lavoro di Excel detta *Modulo*. Un modulo VBA contiene il codice sorgente della macro, ossia l'insieme delle istruzioni della macro stessa. Quando si registra una macro di Excel si può specificare la cartella di lavoro in cui verrà salvata, che può essere:

- La cartella corrente
- Una nuova cartella di lavoro
- La cartella macro personale (File Personal.xls)

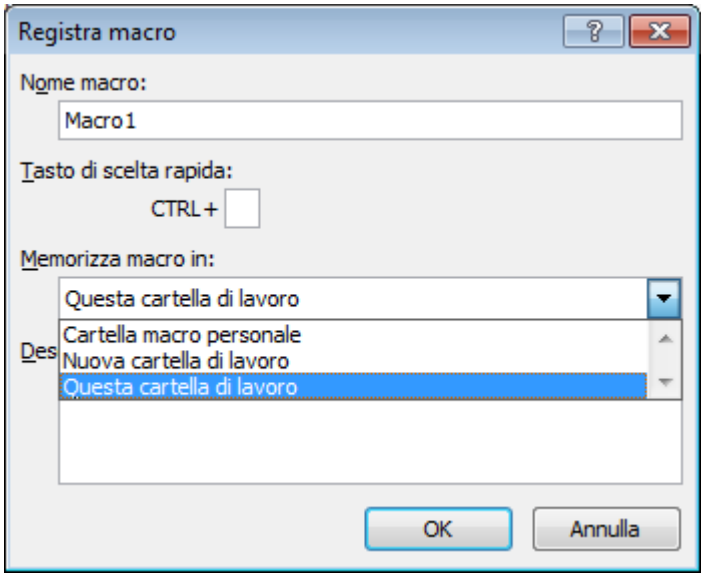


Fig. 1

Excel sceglie, o crea, se necessario, il modulo in cui memorizza la macro registrata e gli assegna il nome di default *Modulo1*, dove **1** è un numero progressivo che indica i moduli che sono stati creati in una cartella di lavoro. Per esempio, la prima volta che si salva una macro registrata nella cartella macro personale (Personal.xls per Excel 2003 e Personal.xlsb per Excel superiore alla 2003) Excel crea un modulo di nome *Modulo1*, se si continua a registrare macro e si salvano sempre nella cartella macro personale, Excel continua a memorizzare le macro nel medesimo Modulo1. Se in seguito si sceglie di memorizzare una macro in una cartella di lavoro diversa, (questa cartella di lavoro) Excel aggiungerà un nuovo modulo denominato sempre Modulo1 in cui memorizzare la macro. Successivamente se si ritorna a memorizzare una macro nella cartella macro personale Excel aggiungerà un nuovo modulo denominato *Modulo2* nella cartella di lavoro *Personal.xls*

Teniamo presente che la cartella macro personale viene caricata all'apertura di Excel pertanto se memorizziamo delle macro in questa cartella esse saranno disponibili per tutte le cartelle di Excel che andremo ad aprire o creare. In pratica se memorizziamo una macro in un foglio di calcolo normale una volta che questo file viene chiuso la macro non sarà più disponibile, mentre se scriviamo una macro che rappresenta una interessante funzione che potrebbe esserci utile in altri file la memorizziamo nel file **Personal.xls** rendendola così disponibile anche per altri file Excel.

Il file Personal.xls non è altro che un normale file di Excel con estensione **xls**, o **xlsb** per versioni di Excel superiori alla 2003, che ha solo la particolarità di essere caricato all'avvio di Excel e di essere un file nascosto, cioè non verrà visualizzato, inoltre quando inseriamo una macro al suo interno, alla chiusura del file di Excel ci viene richiesto il salvataggio del file personale in questo modo

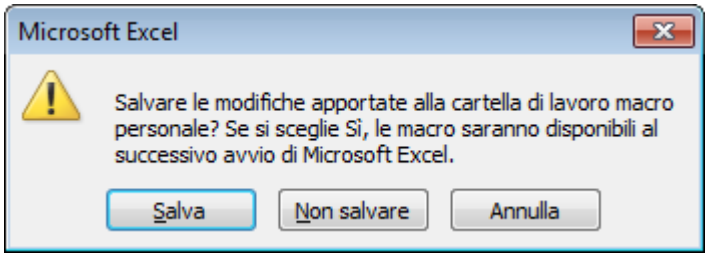


Fig. 2

Se per qualche motivo viene memorizzata una macro nella cartella macro personale e dovesse sorgere qualche problema o interferire con le normali cartelle di lavoro di Excel si deve editare il file *Personal.xls* e cancellare o modificare la macro che presenta le anomalie. Se per qualche motivo non si riesce a modificare la macro e si vuole risolvere rapidamente il problema è possibile ricorrere a maniere drastiche, come la cancellazione del file, senza che la rimozione del file Personal.xls pregiudichi la normale esecuzione di Excel, in quanto se il file Personal.xls è assente nel sistema Excel lo ricrea con le impostazioni di default. A solo scopo informativo il percorso dove viene localizzato il file Personal.xlsb (per Excel 2007 o superiori) è il seguente:

C: \Utenti\NOME UTENTE\AppData\Roaming\Microsoft\Excel\XLStart

Ovviamente dovete prima attivare la visualizzazione dei file nascosti, ma per il momento tralasciamo questo argomento che esula dal contesto della lezione. Per poter vedere i moduli contenuti in un file con estensione xls bisogna ricorrere *all'Editor di Visual Basic*, che è uno strumento che consente di creare moduli, esaminare i contenuti, creare o modificare il codice sorgente delle macro, creare finestre di dialogo e fare tante altre cose relative alla creazione e alla manutenzione di programmi in Visual Basic For Applications.

Per vedere i moduli o il codice sorgente VBA in esso contenuto bisogna avviare l'editor VB e per farlo si può scegliere una di queste opzioni: Selezionare dal menu
Strumenti - Macro - Editor Visual Basic oppure premere semplicemente i tasti **ALT+F11**, Excel avvierà l'editor VB che viene visualizzato come in figura sotto riportata

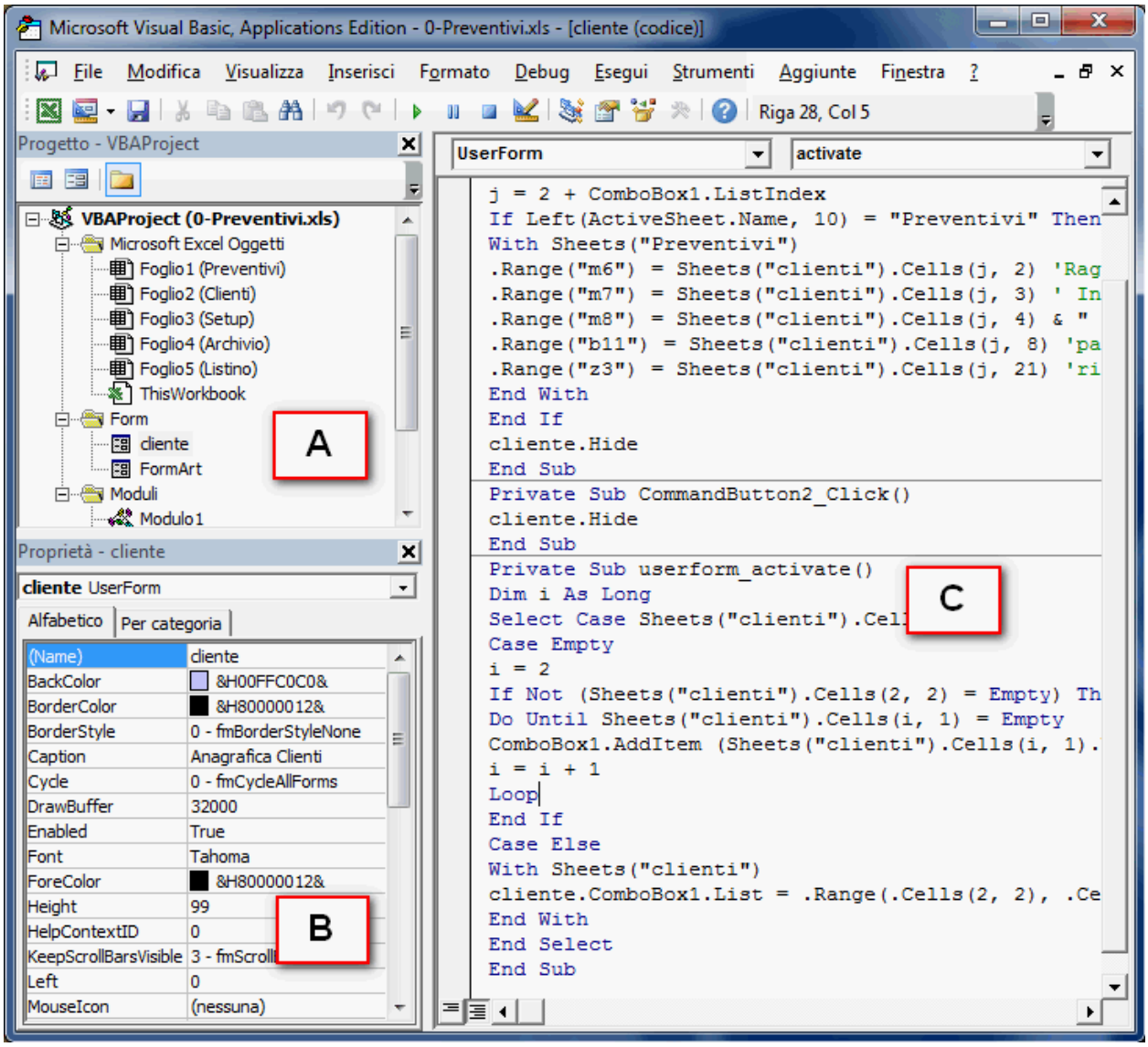


Fig. 3

L'Editor di VB visualizza tre finestre contenute in una finestra principale e ciascuna delle tre finestre visualizza importanti informazioni sul progetto VBA (un progetto è l'insieme di moduli e oggetti memorizzati in una cartella di lavoro). Ognuna delle tre finestre dell'Editor VB viene visualizzata di default nella posizione ancorata come mostrato in **figura 3**. Le tre finestre dell'Editor VB e le relative funzioni sono:

- **Gestione Progetti:** Questa sotto finestra (*Posiz. A Fig. 3*) contiene un diagramma ad albero delle cartelle di lavoro aperte e degli oggetti Excel in esse contenute (oggetti, moduli, riferimenti, Form e così via). Si userà questa finestra per navigare fra i vari moduli e altri oggetti di un progetto VBA

- **Finestra delle Proprietà:** In questa finestra (*Posiz. B Fig. 3*) compare un elenco di tutte le proprietà dell'oggetto attualmente selezionato. La scheda "Alfabetico", della finestra proprietà presenta un elenco ordinato alfabeticamente delle proprietà dell'oggetto selezionato. La scheda per "Categoria" sempre nella stessa finestra le elenca invece ordinate per categoria

- **Finestra del Codice:** La finestra del codice (*Posiz. C Fig. 3*) è quella in cui si può esaminare, modificare o creare ex-novo il codice sorgente VBA. Questa finestra viene usata per scrivere nuove macro o editare macro esistenti

Finestra Gestione Progetti

La finestra "*Gestione progetti*" consente di visualizzare un elenco gerarchico di tutti gli elementi del progetto che sono contenuti nella cartella di lavoro, che includono fogli di lavoro, Forms e moduli. Per default questa finestra utilizza un sistema di visualizzazione definito "*diagramma a albero*". Basta fare clic su uno dei segni **+** che si trovano alla sinistra di una voce del progetto per espandere l'elenco mostrandone la ramificazione. Una volta espanso il diagramma il segno **+** si trasforma in segno **–** e basta cliccare su quest'ultimo per far collassare la ramificazione e chiuderla. La finestra della Gestione progetti contiene altri due pulsanti: *Visualizza codice* e *Visualizza oggetto* evidenziati nella **figura 4** dal rettangolo rosso

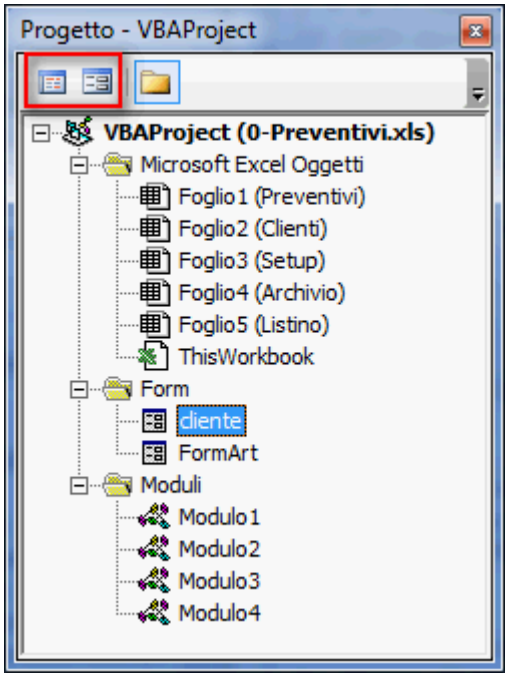


Fig. 4

Il pulsante *Visualizza codice* mostra i contenuti o listato del codice del modulo nella finestra del Codice (*Posiz. C Fig. 3*) mentre il pulsante *Visualizza oggetto* mostra l'oggetto che corrisponde alla voce selezionata, può essere un foglio di calcolo o una Form. Quando si lavora col codice sorgente di una macro nella sotto finestra Codice, si possono anche chiudere le finestre Gestione progetti e Proprietà, per dare più spazio al codice in modo da vederne una parte maggiore.

Si può chiudere quando si vuole una qualsiasi sotto finestra dell'Editor VB (Gestione progetti, Proprietà e Codice) con un clic sul pulsante di chiusura posto nel vertice superiore destro (la classica X) e per visualizzare una delle finestre si può fare clic sul relativo pulsante nella barra strumenti dell'Editor VB evidenziati dal rettangolo rosso in **Figura 4**

Fig. 5

Oppure tramite il percorso dal menù: **Visualizza - Gestione progetti**

La Finestra delle Proprietà

Tutti gli oggetti di Excel hanno delle proprietà che ne controllano l'aspetto e il comportamento, un foglio di lavoro ha la proprietà di essere visibile oppure no, un pulsante ha una proprietà per altezza e larghezza e così via. Per modificare l'aspetto e il comportamento di un oggetto dobbiamo cambiarne le proprietà, e lo possiamo fare selezionando un foglio di lavoro, un modulo o una Form nella finestra *Gestione progetti* e controllare le sue proprietà nella finestra delle proprietà.

La *finestra Proprietà* contiene due schede: *Alfabetico* e *Categoria*. Nella scheda Alfabetico, le proprietà sono visualizzate in ordine alfabetico senza alcun riferimento alla categoria di appartenenza, mentre nella scheda per Categoria, le proprietà sono disposte in ordine alfabetico all'interno di varie categorie.

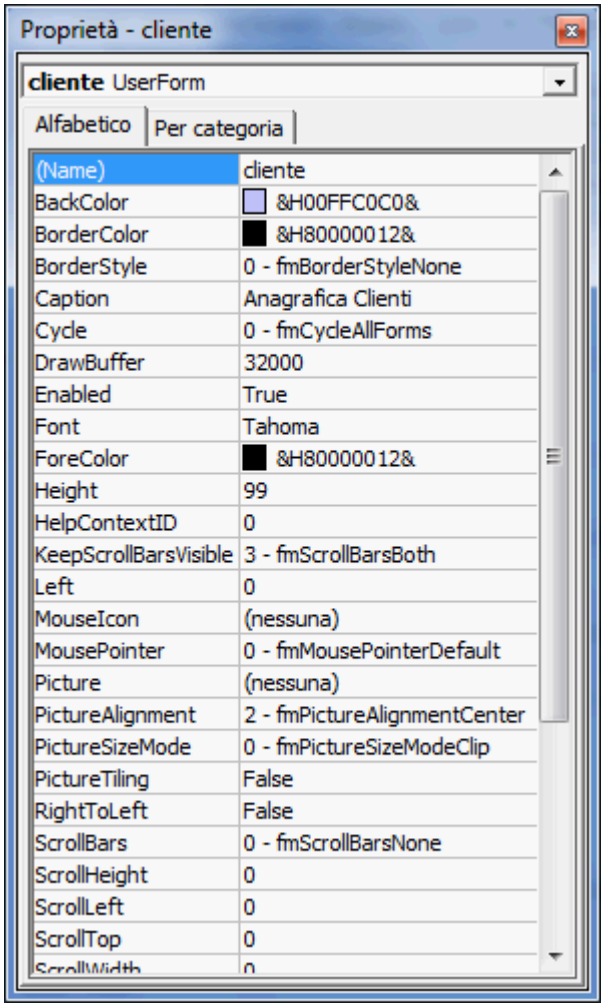


Fig. 6

Le proprietà riportano i valori assegnati agli oggetti che definiscono l'aspetto e le funzionalità degli stessi, solitamente questi valori possono essere letti oppure assegnati, in questo caso si dice che la proprietà è di lettura e scrittura, esistono anche proprietà di sola lettura, alle quali è possibile accedervi in sola lettura. Se la finestra non fosse visibile è possibile accedervi attraverso il percorso dal menu **Visualizza - Finestra proprietà**

La Finestra del Codice

La modalità di visualizzazione di default del codice sorgente VBA è *visualizza modulo intero*, in cui tutto il codice sorgente di un modulo è visualizzato in una finestra di testo che permette lo scorrimento ed ogni macro in questa finestra è separata da una linea continua grigia. L'Editor di VB permette anche di vedere i contenuti di un modulo in modalità Visualizza routine attivabile dalle due icone presenti nel vertice inferiore sinistro della finestra del codice evidenziate dalla freccia rossa in **Figura 7**

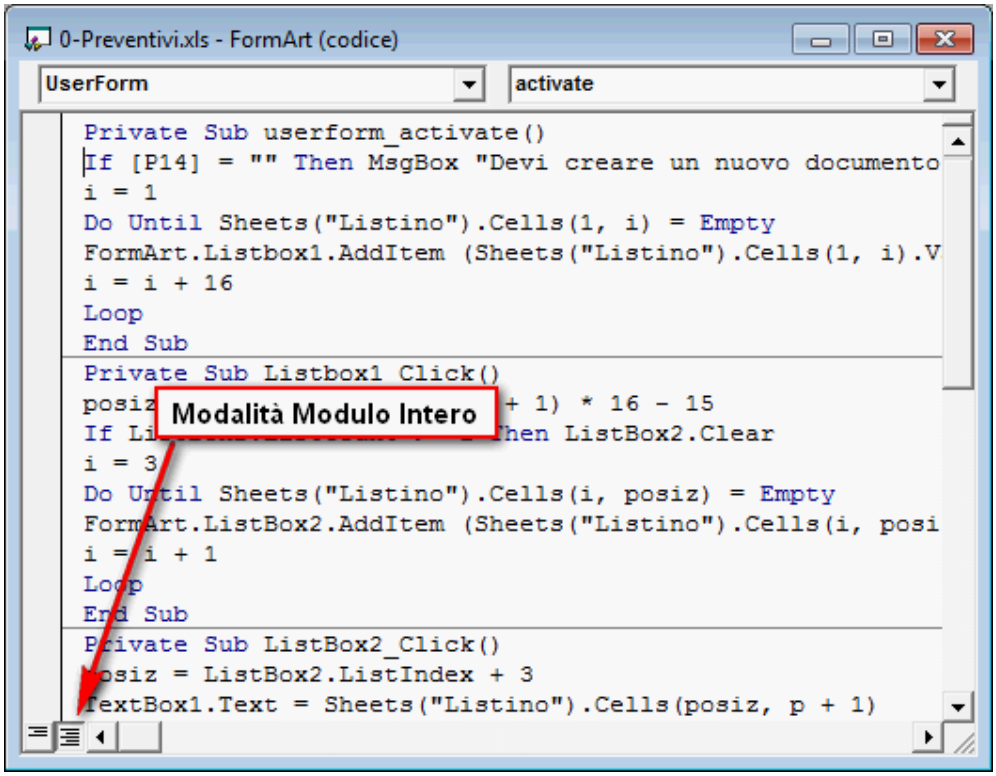


Fig. 7

Quando la finestra Codice è in modalità Routine, come si nota nella **figura 8** si può vedere soltanto il codice sorgente di una macro alla volta

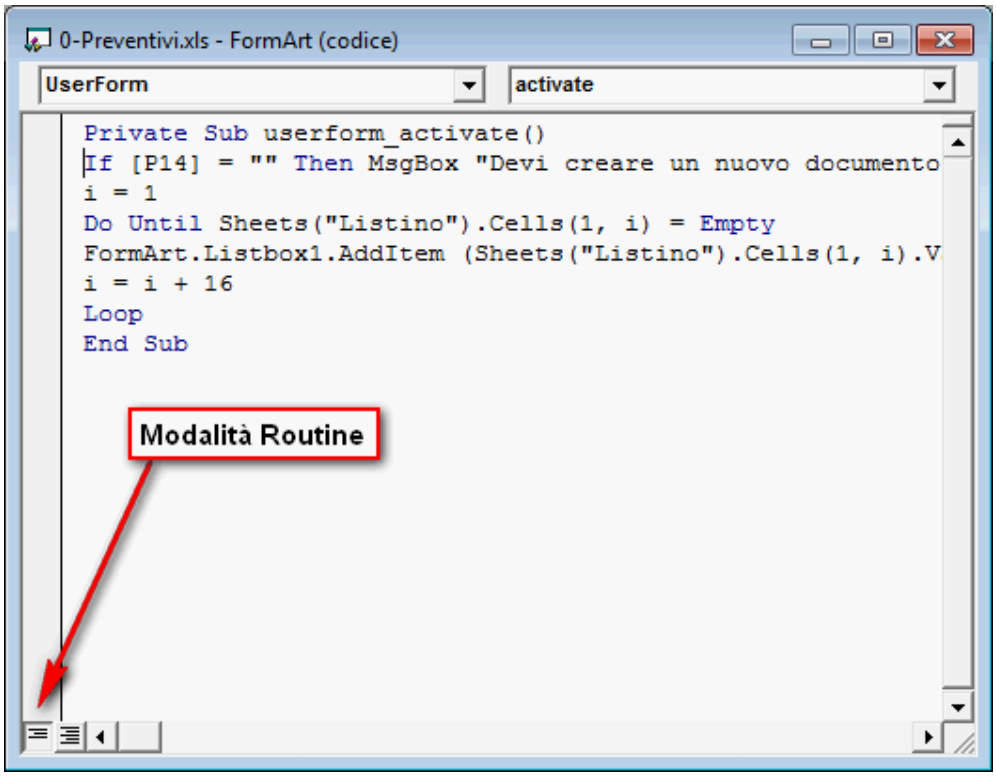


Fig. 8

Per visualizzare una determinata routine nella finestra del codice in modalità Routine basta solo posizionare il cursore col mouse sulla routine e cliccare sull'icona relativa nella **figura 7**, per tornare poi alla visualizzazione a modulo intero si deve cliccare sull'icona a fianco. Nella modalità a Modulo intero si può usare l'elenco a discesa delle procedure per passare rapidamente da una macro all'altra.

Efficienza e prestazioni in VBA

Assieme ai prodotti Office di Microsoft viene distribuito un linguaggio di programmazione, denominato Visual Basic for Application (VBA) ed è utilizzato principalmente per automatizzare le attività, per risparmiare tempo e gestire i dati in modo più efficiente. Usare VBA può causare, a volte, dei rallentamenti all'applicazione principalmente imputabili allo stile di programmazione, infatti è facile cadere in cattive abitudini di programmazione quando si lavora in questo ambiente, sia che si elaborano piccole o ingenti quantità di dati, o che determinate macro vengano richiamate in maniera ricorrente, pertanto è molto importante scrivere un codice snello ed efficiente in modo da ottimizzare le attività riducendone i tempi di esecuzione e migliorando le prestazioni.

Ci sono diversi modi per ottenere ottimi risultati con VBA, lo scopo di questo articolo è quello di illustrare delle semplici regole che indicheranno come migliorare l'efficienza delle cartelle di lavoro, in quanto possono avere un forte impatto sulle prestazioni delle macro.

1. Disattivare il calcolo automatico del foglio di lavoro

Questa regola è nota a molti ed è quella più importante. Quando un nuovo valore viene immesso in una cella del foglio di lavoro, Excel ricalcolerà tutte le celle che fanno riferimento al foglio e se la macro sta scrivendo dei valori nel foglio di lavoro, sarà necessario attendere che venga ricalcolata ogni voce prima di riprenderne il controllo. L'impatto di lasciare il calcolo automatico attivato nel programma può essere una scelta inopportuna e si consiglia vivamente di disattivarlo utilizzando il seguente comando all'inizio della macro: *Application.Calculation = xlCalculationManual*

Se è necessario ricalcolare i valori del foglio di calcolo, mentre la macro è in esecuzione è possibile utilizzare uno dei comandi di seguito descritti, dove il primo ricalcola solo un foglio specifico e il secondo ricalcola solo un intervallo specifico.

```
Worksheets ("Foglio1").Calculate
Range ("A1:A25").Calculate
```

Quando la macro ha terminato la sua esecuzione, il calcolo automatico deve essere ripristinato utilizzando il seguente comando: *Application.Calculation = xlCalculationAutomatic*

2. Disattivare gli aggiornamenti dello schermo

Ogni volta che VBA scrive i dati nel foglio viene aggiornata l'immagine sullo schermo, e questo costituisce un notevole freno alle performance dell'applicazione, per cui è opportuno disattivare questa operazione utilizzando il seguente comando: *Application.ScreenUpdating = FALSE*

Al termine dell'utilizzo della macro è possibile attivare gli aggiornamenti dello schermo con il seguente comando: *Application.ScreenUpdating = TRUE*

3. Lettura e scrittura di dati in una singola operazione

E' fondamentale ridurre al minimo il traffico tra VBA e Excel, specialmente quando è possibile leggere e scrivere dati in blocchi. Ci sono diversi metodi per ottenere questo risultato, un esempio di lettura in un blocco di dati può essere quello di lettura in una matrice. Questo esempio è circa 50 volte più veloce in lettura in ciascuna cella rispetto ad usare un ciclo.

```
Dim myArray() As Variant
myArray= Worksheets("Foglio1").Range("A1:S500").value
```

Allo stesso modo, si possono usare altri metodi, anche se meno efficaci di quello appena proposto come i seguenti:

```
Worksheets("Foglio1").Range("A1:S500").value = myArray
```

```
With Worksheets("Foglio1")
.Range("A1:S500").Value = myArray
End With
```

```
Dim intervallo As Range
Set intervallo = Range("A1:S500")
intervallo.value = myArray
```

4: Dichiarare le variabili

E' meglio evitare di dichiarare una variabile numerica come Variant se non strettamente necessario. Si noti che se si sceglie di utilizzare "Option Explicit" all'inizio della macro qualsiasi variabile indefinita sarà di tipo Variant. Le Variant sono molto flessibili perché possono essere numerico o testo, ma sono lente per l'elaborazione in una formula. Idealmente, è meglio utilizzare variabili Integer o Boolean ove possibile

5: Evitare l'uso eccessivo di funzioni di Excel nel codice

A volte si dà per scontato che funzioni native di Excel sarebbero state efficacemente trattate anche da VBA, ma non è sempre così. Ad esempio, sappiamo che VBA non ha una funzione Max () o Min () e molti utenti utilizzano la funzione di Excel tramite il codice

```
variabile = Application.Max(Value1, Value2)
```

Recentemente ho trovato su internet una versione di una funzione VBA Max () che è 10 volte più veloce rispetto alla funzione di Excel, tuttavia, il codice qui sotto riportato è oltre 80 volte più veloce, pur con la limitazione di avere solo due argomenti e non supporta le matrici, ma il miglioramento in termini di velocità è notevole.

```
Function Max2 (Value1, Value2)
If Value1 > Value2 Then
Max2 = Value1
Else
Max2 = Value2
End If
End Function
```

Suggerisco cautela quando si usano le funzioni di Excel e si dovrebbe sempre valutare l'impatto della riscrittura della funzione. Si noti che qualsiasi comando che inizia con "Application" o "WorksheetFunction" si riferisce a una funzione di Excel

6: Evitare la valutazione Strings

Le variabili Strings (testo) sono lenti da valutare, si consiglia di evitare di valutare stringhe in codice come questo:

```
Select Case Sesso
Case "Maschio"
…… codice
Caso "Femmina"
….. codice
End Select
```

E' bene ricordare che la numerazione assegna un valore numerico costante ad una variabile e VBA è in grado di elaborare i valori enumerati velocemente mantenendo il codice leggibile, inoltre si possono assegnare valori numerici predefiniti o valori specifici per poi essere assegnati in questo modo

```
Public num numSesso
maschio = 0
Femmina = 1
End num

Dim Sesso As numSesso
Select Case Sesso
Case Maschio
... codice
Case Femmina
... codice
End Select
```

Gli operatori booleani sono semplicemente degli switch (interrutori) e possono essere true o false che elaborano molto velocemente. Nell'esempio che segue bMale, bFemale sono variabili booleane. Il codice booleano è di circa 10 volte più veloce rispetto all'utilizzo di stringhe.

```
If bMaschio Then
 ...  codice
ElseIf bFemmina Then
...  codice
End If
```

7: Non selezionare i fogli di lavoro specifici se non necessario

In genere non è necessario utilizzare il comando Select per leggere o scrivere su un foglio di lavoro, non selezionando il foglio si velocizza la procedure di circa 30 volte, evitare questo:

```
Worksheets ("Foglio1"). Select
variabile = Cells (1, 1)
```

Fate questo invece:

```
variabile = Worksheets ("Foglio1"). Cells (1,1)
```

8: Evitare il copia e incolla

Le funzioni di Copia e Incolla (o PasteSpecial) sono lente, si velocizza di circa 25 volte il processo utilizzare il seguente metodo per copiare e incollare valori.

Range("A1:K100").value = Range("A101:K200").value.

Considerazioni finali

Ho trovato utile scrivere una piccola macro per valutare il risparmio di tempo con i vari metodi. La macro esegue semplicemente una funzione un milione di volte e registra il tempo trascorso eseguendo tale metodo. La macro semplice sotto confronta la funzione di Excel Max () per la funzione Max2 indicato nella Regola # 5.

```
'**Valutazione prima funzione
Start_time = Now
For i = 1 To 1000000
value1 = Application.Max(amt1, amt2)
Next i
End_time = Now
Worksheets("sheet1").Cells(1, 2) = End_Time — Start_Time

'**Valutazione seconda funzione
Start_time = Now
For i = 1 To 1000000
value1 = Max2(amt1, amt2)
Next i
End_time = Now
Worksheets("sheet1").Cells(2, 2) = End_Time — Start_Time
```

Esegui macro VBA su un foglio di lavoro protetto

Se si esegue una macro che tenta di apportare delle modifiche in un foglio di lavoro protetto si verifica un errore run-time "Errore di run-time '1004 '. Una possibilità per evitare l'errore è quella di rimuovere la protezione del foglio di lavoro, eseguire il codice, e quindi proteggere nuovamente, come illustrato di seguito:

Foglio1.Unprotect Password: = "123"
Foglio1.Protect Password: = "123"

Tuttavia, questo metodo ha alcuni difetti:

- Se durante l'esecuzione del codice si verifica un errore o viene fermata l'esecuzione della macro, il foglio di lavoro rimarrà senza protezione
- La password di protezione del foglio di lavoro verrà visualizzato nel codice VBA e anche se è possibile proteggere il progetto VBA per impedire l'accesso o la visualizzazione del codice, in rete esistono tantissimi decompilatori che rendono vana la protezione della password
- Sarà necessario inserire il codice (Protect e Unprotect) più volte in ciascuna macro.

Per superare il primo inconveniente in cui il foglio rimane sprotetto sul codice di errore, è possibile utilizzare un *ErrorHandler*, come illustrato di seguito:

```
Sub protezione1 ()
Foglio1.Unprotect Password: = "123"
On Error GoTo ErrHandler
Foglio1.Cells (1, 1) = UCase ("Ciao Mondo")
Foglio1.Cells (2, 1) = 5/0
Foglio1.Cells (3, 1) = Application.Max (24, 112, 66, 4)
Foglio1.Protect Password: = "123"
ErrHandler: Foglio1.Protect Password: = "123"
End Sub

Sub protezione2 ()
Foglio1.Unprotect Password: = "123"
On Error Resume Next
Foglio1.Cells (1, 1) = LCase ("CIAO MONDO")
Foglio1.Cells (2, 1) = 5/0 .
Foglio1.Cells (3, 1) = Application.Min (24, 112, 66, 4)
Foglio1.Protect Password: = "123"
On Error GoTo 0
End Sub
```

Dichiarazione On Error

On Error Resume Next: Specifica che quando si verifica un errore di run-time, il controllo passa all'istruzione immediatamente successiva all'istruzione in cui si è verificato l'errore stesso e l'esecuzione continua da quel punto.

On Error GoTo 0: Disattiva l'intercettazione degli errori
On Error GoTo Line: Attiva la routine di gestione degli errori che inizia alla riga specificata. On Error GoTo intercetta tutti gli errori.

Un modo per eseguire le macro in un foglio di lavoro protetto potrebbe essere quella di utilizzare l'argomento *UserInterfaceOnly* nel metodo Protect, impostando l'argomento UserInterfaceOnly a True, nella maniera:

Foglio1.Protect Password: ="123", UserInterfaceOnly: = True.

UserInterfaceOnly è un argomento facoltativo nel metodo Protect e il suo valore predefinito è *False*. Impostarlo a True significa che la protezione del foglio di lavoro vale solo per l'interfaccia utente e non si applica alle macro e questo permetterà di eseguire tutte le macro nel foglio di lavoro. Se si omette questo argomento, la protezione si applica sia alle macro che all'interfaccia utente.

Si può osservare che se si applica il metodo Protect con l'argomento UserInterfaceOnly impostata a True per un foglio di lavoro e quindi si salva la cartella di lavoro, l'intero foglio di lavoro, non solo l'interfaccia, sarà completamente protetto quando si riapre la cartella di lavoro. Per riattivare la protezione di interfaccia utente dopo che la cartella di lavoro è aperta, è necessario ancora una volta applicare il metodo Protect con UserInterfaceOnly impostata su true.

Per riattivare la protezione di interfaccia utente dopo che la cartella di lavoro è aperta, è possibile utilizzare l'argomento UserInterfaceOnly nell'evento Open in cui esso viene attivato in tutti i fogli di lavoro o in quelli specificati, ogni volta che la cartella di lavoro è aperta. È inoltre possibile utilizzare l'argomento UserInterfaceOnly in un foglio di lavoro, all'inizio della macro, per abilitare la protezione di interfaccia utente ogni volta che la macro viene eseguita. Vedere esempi riportati di seguito:

```
Sub protezione3 ()
Foglio1.Protect Password: = "123", UserInterfaceOnly: = True
Foglio1.Cells (1, 1) = UCase ("Ciao Mondo")
End Sub
```

Se tutti i fogli usano la stessa password, è possibile impostare l'argomento UserInterfaceOnly come TRUE come segue. Notare che il codice è inserito in ThisWorkbook della cartella di lavoro

```
Private Sub Workbook_Open ()
Dim ws As Worksheet
For Each ws In ThisWorkbook.Worksheets
ws.Protect password: = "123", UserInterfaceOnly: = True
Next ws
End Sub
```

Per abilitare la protezione a livello di interfaccia utente solo se i fogli di lavoro hanno password differenti in fogli di lavoro specifici, impostare l'argomento UserInterfaceOnly come True nell'evento Open, come segue.

```
Private Sub Workbook_Open ()
Foglio1.Protect Password: = "123", UserInterfaceOnly: = True
Foglio2.Protect Password: = "456", UserInterfaceOnly: = True
End Sub
```

Se si desidera eseguire una macro su un foglio di lavoro protetto, e mantenere il codice visibile e modificabile nell'editor di VB ma non si vuole rivelare o visualizzare la password si può utilizzare il seguente metodo.

Se per esempio si utilizzano tre fogli di lavoro (Foglio1, Foglio2 e Foglio3) in una cartella di lavoro che sono protetti da password, si può inserire un foglio di lavoro supplementare (Foglio4) e renderlo 'molto nascosto' (*VeryHidden*), il che significa che non sarà visibile all'utente e può essere visibile solo con un codice VBA, perché non compare nell'elenco Scopri dell'interfaccia. Un foglio può essere 'molto nascosto' utilizzando una singola linea di codice VBA come la seguente

```
Sub foglio_nascosto()
Worksheets ("Foglio4"). Visible = xlVeryHidden
End Sub
```

E' possibile inserire una password, per visualizzare il foglio 'molto nascosto', il che significa che la password dovrà essere ricordato separatamente e non sarà visibile o accessibile finché il foglio è veryhidden. Però è possibile memorizzare le password che proteggono i fogli di lavoro su cui si desidera eseguire le macro, e queste macro richiameranno le password memorizzate nel foglio "VeryHidden"

```
Sub foglio_nascosto()
Dim pswd As String
pswd = Cells (1, 1)
mypass = pswd
pswdMatch = InputBox ("Inserire la password per visualizzare il foglio")
If pswdMatch = pswd Then
Worksheets ("Foglio4"). Visible = True
Else
Exit Sub
End If
End Sub

Sub Protezione5()
Dim passfgl2 As String
passfgl2 = Foglio4.Cells (2, 1)
Foglio2.Protect Password: = passfgl2 , UserInterfaceOnly: = True
Foglio2.Cells (1, 1) = UCase ("Ciao Mondo")
End Sub
```

I 3 esempi esposti poco sopra faranno in modo di eseguire la macro su un foglio di lavoro protetto, la cui password è nascosta nel Foglio4 e lo stesso è stato protetto e reso "molto nascosto". E' possibile abilitare il *Filtro automatico* quando la protezione dell'interfaccia utente è attiva. Questa proprietà di attivare il filtro non viene salvata con il foglio di lavoro Quando la protezione dell'interfaccia utente è attiva si può utilizzare la proprietà AutofilterMode tramite VBA per utilizzare il filtro automatico si può utilizzare una macro in questo modo

```
Sub attiva_filtro1 ()
With Foglio1
. EnableAutoFilter = True
.Protect Password:="123" , contents:=True, UserInterfaceOnly:=True
. AutoFilterMode = False
.Range("A1:B10" ).AutoFilter Field:=1, Criteria1:= "<35"
End With
End Sub

Sub attiva_filtro2 ()
With Foglio1
.Protect Password:="123" , contents:=True, UserInterfaceOnly:=True
.Range("A1:B10" ).AutoFilter
.Range("A1:B10" ).AutoFilter Field:=1, Criteria1:= ">35"
End With
End Sub
```

Inoltre è possibile utilizzare UserInterfaceOnly per consentire il raggruppamento in un foglio di lavoro protetto quando l'interfaccia utente ha la protezione attiva

```
Sub attiva_gruppo()
With Foglio1
. EnableOutlining = True
.Protect Password:="123" , contents:=True, UserInterfaceOnly:=True
.Rows ("12:14"). Group
End With
End Sub
```

Metodo Protezione Worksheet senza UserInterfaceOnly

Questo metodo Protegge un foglio di lavoro in modo che non può essere modificato. Se sono necessarie modifiche da apportare a un foglio protetto, è possibile utilizzare il metodo Protect di un foglio protetto se viene fornita la password. Inoltre, un altro metodo sarebbe quello di rimuovere la protezione del foglio di lavoro, apportare le modifiche necessarie, e quindi proteggere nuovamente il foglio di lavoro. Vediamo alcuni esempi:

```
Sub consenti_ordinamento1 ()
Foglio1.Unprotect Password: = "123"
Foglio1.Range. ("G1: H10") Locked = False
Foglio1.Protect Password: = "123", contents: = True, AllowSorting: = True
Foglio1.Range. ("G1: H10") Sort Key1: = Sheet1.Range ("G1"), Order1: = xlDescending
End Sub

Sub consenti_formattazione1 ()
Foglio1.Protect Password: = "123", AllowFormattingCells: = True
Foglio1.Range. ("G1: H10") Font.Bold = True
Foglio1.Range. ("A1: B10") Font.Color = vbBlue
End Sub

Sub consenti_formattazione2 ()
Foglio1.Protect Password: = "123", AllowFormattingColumns: = True,
AllowFormattingRows: = True
Foglio1.Columns (2). ColumnWidth = 25
Foglio1.Rows ("4:5"). RowHeight = 25
End Sub
```

Gestione degli Eventi nel foglio di lavoro

Ci sono molti modi per eseguire una macro, possiamo "lanciarla" avviando la finestra di dialogo Macro, tramite un tasto di scelta rapida, un comando di menu o un pulsante personalizzato, la caratteristica comune di tutti questi metodi è che per eseguire la routine l'utente deve fare qualcosa, scegliere un comando, premere una combinazione di tasti o fare clic su un pulsante. In VBA vi sono tuttavia diverse tecniche che permettono di eseguire delle routine automaticamente quando si verifica un determinato evento, ad esempio l'apertura di una cartella di lavoro o l'apertura di un determinato foglio di lavoro.

Vediamo adesso come si creano routine per eventi specifici associati agli oggetti di Excel, prima di tutto però può essere utile rivedere alcuni concetti relativi alle routine di evento. L'esecuzione di una routine guidata dagli eventi è ciclica, anziché seguire un percorso lineare dall'inizio alla fine, la routine rimane in attesa che si verifichino determinati eventi e quindi rispondere a questi. Un evento è qualcosa che avviene nel programma, per esempio l'apertura di una cartella di lavoro, l'attivazione di un foglio di lavoro, il salvataggio di una cartella di lavoro. Poiché gli eventi specifici che si verificano durante l'esecuzione del programma ne determinano il comportamento, si dice che questo è guidato da tali eventi. Quando si verifica un evento il programma esegue una o più routine correlate a tale evento.

Eventi del foglio di lavoro

Gli oggetti di Excel quali *Workbook*, *Worksheet* contengono un modulo di classe in cui sono memorizzate le routine di gestione degli eventi di tali oggetti. Il modulo di classe di un foglio di lavoro di Excel (Worksheet) si raggiunge cliccando sul nome di un foglio, come si vede in *figura 1* indicato dalla *freccia rossa*e selezionando l'oggetto Worksheet nell'apposita finestra a discesa indicata dalla *freccia blu*, compare nella finestra del codice la prima routine degli eventi, oltre alla quale, ne sono presenti molti altri selezionabili nella finestra a discesa indicata dalla *freccia verde*. Quando selezionate un evento nell'elenco routine (freccia verde), VBA inserisce una dichiarazione vuota per quella routine. Nella finestra Codice della figura 1 potete vedere la dichiarazione vuota della routine di gestione dell'evento **SelectionChange** del foglio di lavoro

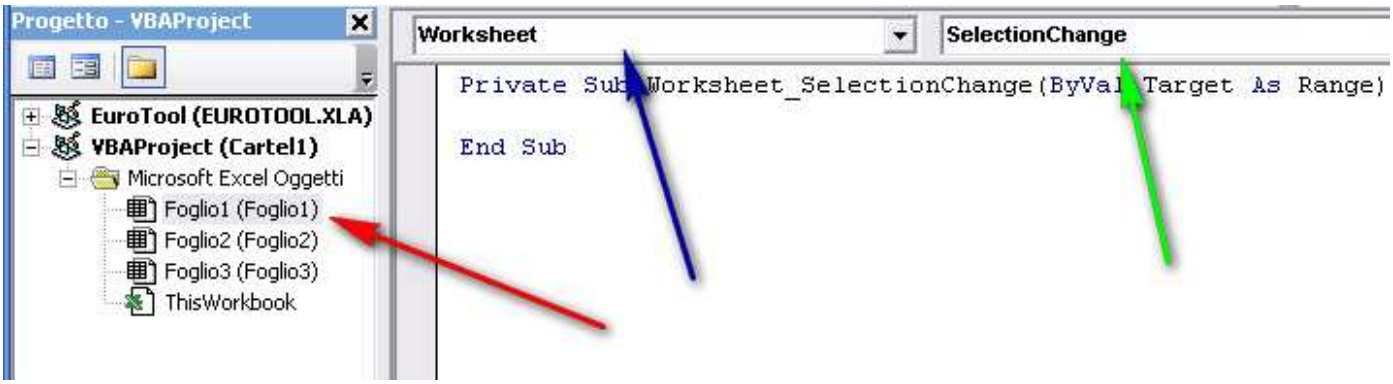


Fig. 1

I vari eventi associati all'oggetto Worksheet sono:

Private Sub Worksheet_Activate ()

Questo evento viene innescato quando il foglio viene attivato e se al suo interno è presente del codice VBA, questo viene eseguito. Alcuni esempi di utilizzo di questo evento

```
Private Sub Worksheet_Activate()
   UserForm1.Show 'mostra la userform1
End Sub

Private Sub Worksheet_Activate()
 Range("B1:B20").Sort Key1:=Range("B1"), Order:=xlAscending 'ordinare il range B1:B20
End Sub

Private Sub Worksheet_Activate()
   MsgBox "Hai attivato il Foglio1" 'avviso che è stato attivato il foglio1
End Sub
```

Private Sub Worksheet_SelectionChange (ByVal Target As Range)
Questo evento si verifica quando si seleziona una cella nel foglio ed il parametro Target corrisponde alla cella selezionata.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
 'selezionare la cella A1 se è vuota
    If Range("A1") = "" Then Range("A1").Select
End Sub

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
 'verificare se la cella selezionata si trova nel Range B5: E20
    Dim cella1 As Range
        If Target.Cells.Count > 1 Then
           MsgBox "Seleziona una sola cella"
           Exit Sub
        End If
    Set cella1 = Range("B5:E20")
    If Application.Intersect(Target, cella1) Is Nothing Then
        MsgBox "Fuori Range"
    Else
        MsgBox "Nel Range"
    End If
End Sub

Option Explicit
 'intercettare il cambiamento del colore di sfondo nelle celle
Dim x As Integer, Cell As String
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    On Error Resume Next
    If Cell = "" Then
        x = Target.Interior.ColorIndex
        Cell = Target.Address
        Exit Sub
    End If
    If Range(Cell).Interior.ColorIndex <> x Then _
        MsgBox "Il Colore della cella " & Cell & " è cambiato"
    x = Target.Interior.ColorIndex
    Cell = Target.Address
End Sub
```

Private Sub Worksheet_Change (ByVal Target As Range)
Questo evento viene generato quando il contenuto di una cella nel foglio di lavoro viene modificato, la procedura non tiene conto dei cambiamento di formato nella cella, inoltre il parametro Target corrisponde alla cella modificata.

```
Private Sub Worksheet_Change(ByVal Target As Range)
 'avvisa che si sta modificando una cella
    If Target.Count > 1 Then Exit Sub
    MsgBox "Vuoi Modificare la cella " & Target.Address & "  col valore   " & Target.Value
End Sub

Private Sub Worksheet_Change(ByVal Target As Range)
 'converte in maiuscolo le celle modificate nel ranhe A1:A10
    If Intersect(Target, Range("A1:A10")) Is Nothing Or Target.Cells.Count > 1 Then Exit Sub
    Application.EnableEvents = False
    Target.Value = UCase(Target.Value)
    Application.EnableEvents = True
End Sub

Private Sub Worksheet_Change(ByVal Target As Range)
 'cambia il colore del font della cella se viene modificata
    Target.Font.ColorIndex = 5
End Sub
```

Private Sub Worksheet_Deactivate ()
Questo evento viene attivato quando il foglio è spento (commutazione tra i fogli della stessa cartella di lavoro.) La procedura non viene avviato se si attiva un'altra applicazione o un'altra cartella di lavoro di Excel.

```
Private Sub Worksheet_Deactivate()
 'proteggere un foglio all'uscita
    ActiveSheet.Protect DrawingObjects:=True, Contents:=True, Scenarios:=True
End Sub

Private Sub Worksheet_Deactivate()
 'nascondere un foglio
    Sheets("Foglio3").Select
    ActiveWindow.SelectedSheets.Visible = False
End Sub

Private Sub Worksheet_Deactivate()
 'avvisare che è stato abbandonato un foglio
   MsgBox "Hai disattivato " & Name & "." & vbCrLf & "e sei passato nel " & ActiveSheet.Name
& "."
End Sub
```

Private Sub Worksheet_BeforeDoubleClick (ByVal Target As Range, Cancel As Boolean)
Identifica il doppio click in una cella e il parametro Target corrisponde alla cella che riceve il doppio click, mentre il parametro Cancel disabilita l'azione della macro associata con un evento. Il doppio click consente di modificare la cella, ma se si specifica il valore di Cancel = True la modifica verrà impedita.

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
 'restituire l'indirizzo della cella che ha ricevuto il doppio click
  MsgBox "Hai fatto Doppio click sulla cella " & Target.Address
   Cancel = True
End Sub

Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
 'attivare il foglio 1 facendo doppio click
  If Target.Address = "$A$1" Then Worksheets("Foglio1").Activate
End Sub

Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
 'restituisce l'indirizzo di riga e Colonna dove si è fatto il doppio click
    Select Case Target.Column
        Case 1   'colonna A
            MsgBox "Hai fatto doppio click nella Colonna A e nella riga  " & Target.Row
        Case 2   'colonna B
            MsgBox " Hai fatto doppio click nella Colonna A e nella riga " & Target.Row
        Case 3   'colonna C
            MsgBox " Hai fatto doppio click nella Colonna A e nella riga " & Target.Row
        Case Else
            MsgBox " Hai fatto doppio click nella Colonna A e nella riga D o altre"
    End Select
End Sub
```

Private Sub Worksheet_BeforeRightClick (ByVal Target As Range, Cancel As Boolean)
Questo evento si verifica quando si utilizza il pulsante destro del mouse su una cella nel foglio, il parametro Target corrisponde alla cella che riceve il tasto destro del mouse, mentre il parametro Cancel disabilita l'azione della macro associata a questo evento.

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range,Cancel As Boolean)
 'nasconde le colonne A-B e C cliccando col destro sulla cella D5
  If Target.Address(0, 0) = "D5" Then
    With Columns("A:C").EntireColumn
      .Hidden = Not .Hidden
      Target.Value = IIf(.Hidden, "Colonne", "hide") & " Nascoste"
    End With
  End If
  Cancel = True
End Sub

Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
 'impedisce di copiare il contenuto del range
MsgBox "Contenuto protetto dalla copia"
Cancel = True
If Not Application.Intersect(Target, Range("A1:C20")) Is Nothing Then Exit Sub
End Sub
```

Private Sub Worksheet_Calculate ()
Questo evento viene attivato quando il foglio di lavoro viene ricalcolato solo se l'opzione di calcolo automatico non è attivata.

```
Private Sub Worksheet_Calculate()
 'regolare le dimensioni delle colonne da A a F ogni volta che si ricalcola il foglio
 Columns("A:F").AutoFit
End Sub

Private Sub Worksheet_Calculate()
    If IsNumeric(Range("A1")) Then
      If Range("A1").Value >= 100 Then
        MsgBox "Il Range A1 ha raggiunto il limite 100", vbInformation
      End If
    End If
End SubPrivate Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
 'impedire la comparsa del menu pop-up del destro del mouse
    Cancel = True
End Sub
```

Private Sub Worksheet_FollowHyperlink (ByVal Target As Hyperlink)
L'evento si verifica quando un collegamento viene attivato nel foglio di lavoro, il parametro Target è il collegamento ipertestuale

```
Private Sub Worksheet_FollowHyperlink(ByVal Target As Hyperlink)
 'visualizzare l'indirizzo del link che hai appena fatto clic.
    MsgBox Target.Address & vbCrLf & Target.SubAddress
End Sub

Private Sub Worksheet_FollowHyperlink(ByVal Target As Hyperlink)
 'mantiene una lista, di tutti i collegamenti che sono stati visitati dal foglio di lavoro attivo.
    With UserForm1
        .ListBox1.AddItem Target.Address
        .Show
    End With
End Sub
```

Private Sub Worksheet_PivotTableUpdate (ByVal Target As pivot)
Questo evento si verifica quando si aggiorna una tabella Pivot contenuta nella scheda, il parametro Target è la tabella pivot.

```
Private Sub Worksheet_PivotTableUpdate(ByVal Target As PivotTable)
    MsgBox "La tabella '" & Target.Name & "' è stata aggiornata"
End Sub

Private Sub Worksheet_PivotTableUpdate(ByVal Target As PivotTable)
  MsgBox "La connessione della tabella Pivot è stata aggiornata"
 End Sub
```

Gestione Eventi di foglio di lavoro con Un Modulo Di Classe

Quando si utilizzano dei controlli in un foglio di lavoro, spesso si ha la necessità di utilizzare gli eventi per gestire le azioni che possono essere eseguite. Al tempo stesso se si usano diversi controlli, può diventare problematico far interagire i vari controlli in quanto si deve aggiungere una o più sub evento per ognuno di essi. Per semplificarne la programmazione si può usare un modulo di classe con una singola routine evento per una serie di controlli uguali

Possiamo capirne meglio il significato vedendo un esempio partendo da un foglio di lavoro in cui inseriamo due serie di pulsanti di opzione tramite il percorso dal menu **Sviluppo – Inserisci** e scegliamo il pulsante di Opzione dalla sezione Controlli ActiveX

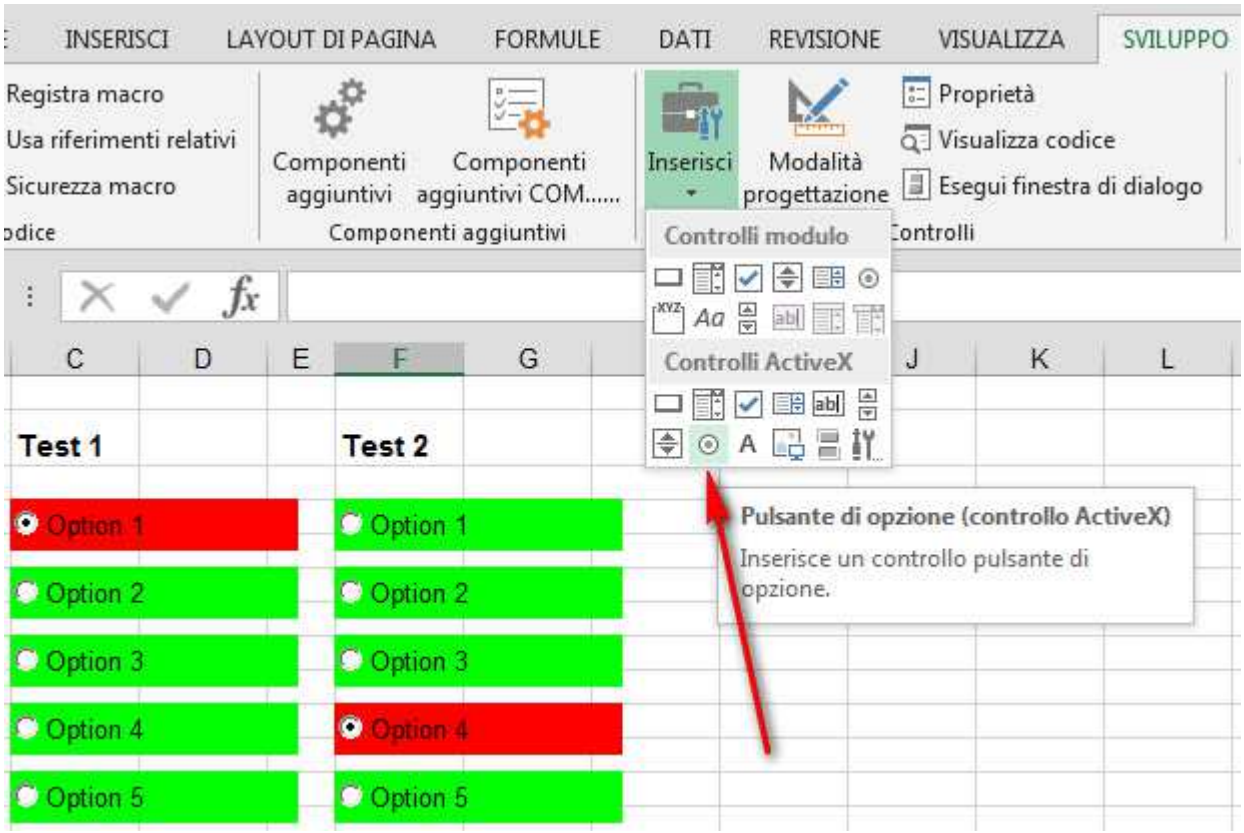


Fig. 1

Una volta inseriti e allineati i vari controlli, per poter farli lavorare come due gruppi separati di pulsanti, dovremmo assegnare due **GroupName** diversi, rispettivamente **Test1** e **Test2** tramite la modalità **Progettazione** e la finestra delle proprietà che troviamo nella scheda **Sviluppo**

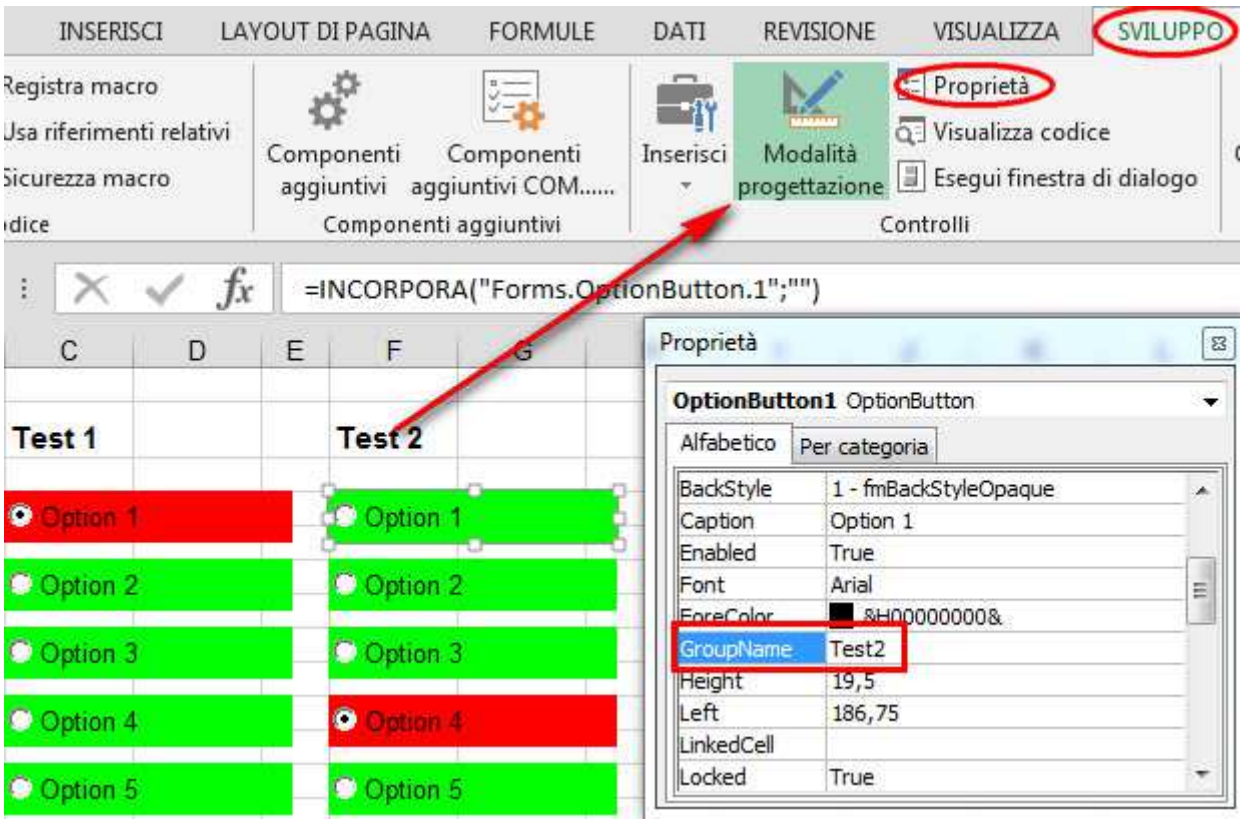


Fig. 2

In questo esempio faremo in modo che il pulsante di opzione selezionato diventi rosso mentre quelli non selezionati saranno di colore verde, inoltre mandiamo anche un avviso a video che riporti il nome del pulsante selezionato

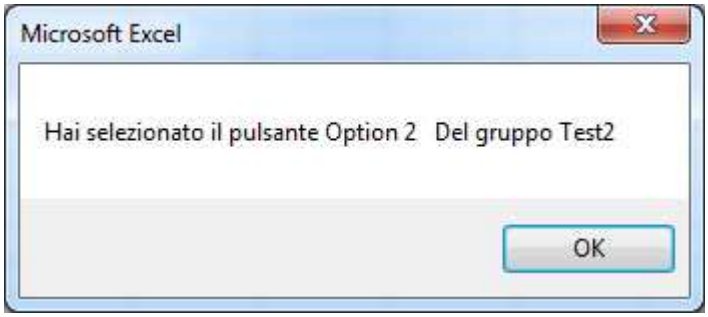


Fig. 3

Iniziamo inserendo un modulo di classe, entrando nell'editor di VBA dal menu **Inserisci – Modulo di classe**. Una volta inserita selezioniamo la classe e nella finestra delle proprietà, cambiamo il suo nome predefinito da Class1 a **clsTest**

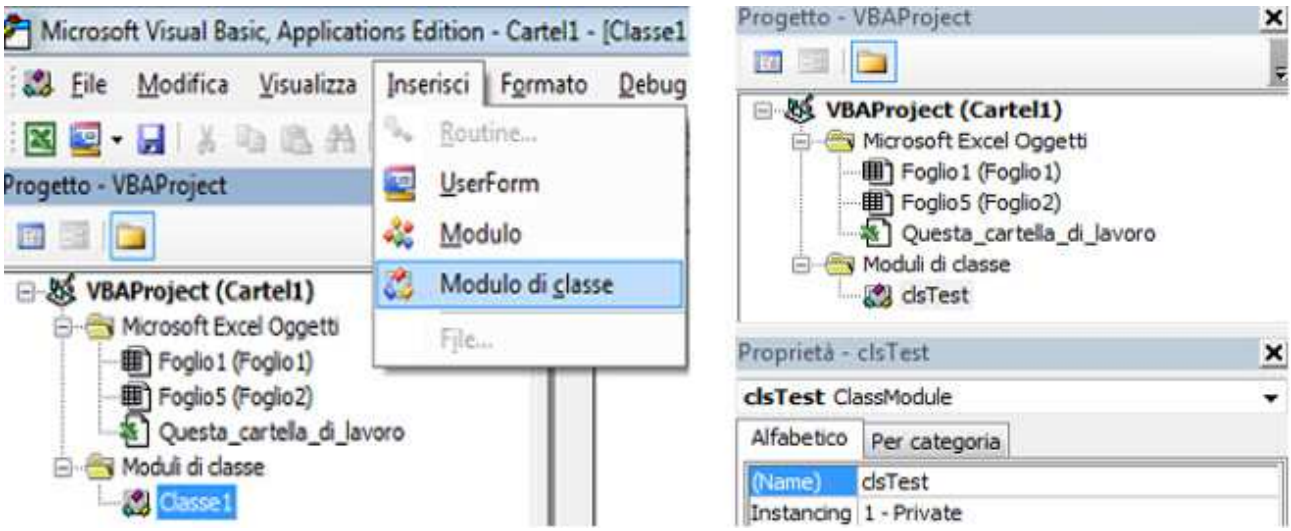


Fig. 4

Ora, nella finestra del codice digitiamo queste righe:

```
Option Explicit
Private WithEvents testOpt As MSForms.OptionButton
```

Tramite la parola chiave **WithEvents** possiamo specificare che una variabile dichiarata (testOpt) si riferisca a un'istanza di una classe in grado di generare eventi. Infatti dopo aver fatto questo, si sarà in grado di selezionare la variabile testOpt dall'elenco a discesa nella parte superiore sinistra della finestra del codice

Fig. 5

Una volta selezionata, per impostazione predefinita, nella finestra del codice appare l'evento click in questo modo

```
Private Sub testOpt_Click()

End Sub
```

Se ora si fa clic sul menu a discesa a destra, verranno elencati tutti gli eventi disponibili per questo tipo di controllo attraverso il modulo di classe e selezioniamo l'evento **Change** dal menu a discesa e rimuoviamo l'evento click

Fig. 6

E otterremo qualcosa di simile:

```
Option Explicit
Private WithEvents testOpt As MSForms.OptionButton

Private Sub testOpt_Change()

End Sub
```

Si diceva di cambiare il colore del controllo selezionato e, naturalmente, anche cambiare il colore del controllo deselezionato, inoltre si voleva anche un messaggio che avvisi quale controllo è stato selezionato. A tal proposito ho pensato di usare questo codice nel modulo di classe

```
Private Sub testOpt_Change()
    If testOpt.Value = 0 Then
        testOpt.Object.BackColor = RGB(0, 255, 0)
    Else
        MsgBox "Hai selezionato il pulsante " & testOpt.Caption & "   Del gruppo " &
testOpt.GroupName
        testOpt.Object.BackColor = RGB(255, 0, 0)
    End If
End Sub
```

Quando si clicca su un pulsante di opzione per selezionarlo, sia il pulsante selezionato e i pulsanti deselezionati eseguiranno i loro eventi di modifica, quindi questa Sub Evento sarà eseguita due volte, una volta per il controllo selezionato e una volta per il controllo deselezionato, il primo avrà valore 1 e il secondo valore 0. Tutto ciò che deve essere fatto ora è quello di collegare i controlli sul foglio di lavoro al modulo di classe. Per prima cosa si deve scrivere del codice nel modulo di classe che riceverà l'oggetto che sta per "ascoltare" da una routine di inizializzazione

```
Public Property Set Control(obtNew As MSForms.OptionButton)
    Set testOpt = obtNew
End Property
```

E infine distruggiamo la classe per liberare risorse

```
Private Sub Class_Terminate()
    Set testOpt = Nothing
End Sub
```

Ora abbiamo bisogno di creare le istanze di questo modulo di classe per quanti controlli abbiamo da collegare. Inseriamo un modulo normale nel progetto e useremo una variabile Collection per contenere le istanze dei moduli di classe:

```
Dim testC As Collection
```

Poi con un ciclo scorriamo tutti gli oggetti del foglio di lavoro e agganciamo i controlli OptionButton all'evento, ecco il codice per l'intero modulo:

```
Option Explicit
Dim testC As Collection

Sub InitializeEvents()
    Dim ctrOpt As OLEObject
    Dim Fgl As Worksheet
    Dim clsEvents As testOpt
    Set Fgl = ThisWorkbook.Worksheets(1)
    If testC Is Nothing Then
        Set testC = New Collection
    End If
    'Ciclo attraverso tutti i controlli
    For Each ctrOpt In Fgl.OLEObjects
        If TypeName(ctrOpt.Object) = "OptionButton" Then
            'Creiamo una nuova istanza della classe
            Set clsEvents = New testOpt
            Set clsEvents.Control = ctrOpt.Object
             'Aggiungiamo l'istanza ad una collezione
            testC.Add clsEvents
        End If
    Next
End Sub

Sub TerminateEvents()
    'distruggiamo la classe per liberare memoria
    Set testC = Nothing
End Sub
```

Il passo finale è quello di assicurarsi che il codice della routine **InitializeEvents** venga eseguito all'apertura della cartella, per cui useremo l'evento Workbook_Open nel modulo ThisWorkbook inserendo questo codice

```
Option Explicit

Private Sub Workbook_Open()
    InitializeEvents
End Sub
```

Infine, quando la cartella di lavoro viene chiusa o se si vuole fermare l'esecuzione della classe di rispondere agli eventi, è necessario distruggere la classe per liberare risorse con questo codice

```
Private Sub Class_Terminate()
    Set testC = Nothing
End Sub
```

Da notare che la routine **Class_Terminate** verrà eseguita per ciascuna istanza della classe creata con la routine **InitialiseEvents**. Ovviamente questa sub deve essere eseguita quando la cartella di lavoro si chiude, quindi nel modulo ThisWorkbook, aggiungere:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    TerminateEvents
End Sub
```

Gestire gli input da tastiera in un controllo TextBox

Durante la stesura di programmi o verticalizzazioni in VBA si verifica molto spesso la necessità di fare in modo che un utente in una TextBox possa digitare<u>solo</u> un input specifico, come numeri interi, stringhe etc. sia per evitare errori di elaborazione del dato inserito che per una corretta scrittura di dati specifici nei relativi fogli di lavoro. E' possibile eseguire il controllo sul testo immesso anche successivamente all'immissione, ma risulta molto più comodo limitare la scelta del tipo di dati da inserire (numeri, stringhe etc.) direttamente nella TextBox utilizzando l'evento "*KeyPress*", che contiene il tasto digitato, e il suo codice Ascii per controllarlo. Prima di iniziare ad usare codice VBA per programmare il TextBox è consigliato stilare una lista delle opzioni e dei vincoli che la casella di testo (TextBox) deve avere, inoltre si deve aggiungere anche il nostro obiettivo finale, che è quello di scrivere il valore desiderato in una cella, in formato numerico, nel foglio di lavoro, pertanto la lista potrebbe essere come la seguente:

1. Accettare solo numeri, una virgola o un segno – (meno)
2. Accettare solo un segno meno, che si deve trovare all'inizio della stringa
3. Accettare solo un punto posizionato nella posizione desiderata.
4. Scrivere il valore del TextBox in un formato numerico nel foglio di lavoro

Infine, ci si deve porre una domanda significativa; *il nostro codice viene usato una sola volta oppure poco frequentemente o viene richiamato ripetutamente?* Se è così, possiamo costruire una funzione e richiamarla ogni volta che ne abbiamo bisogno.

Iniziamo con la costruzione di una UserFom denominata "*Form1*", in cui riponiamo una TextBox denominata "*Text1*" e un pulsante di comando denominato "*Command1*" che cliccando su di esso convaliderà l'assegnazione del valore della nostra TextBox nell'apposita cella del foglio di lavoro. Iniziamo con la verifica del testo inserito nella TextBox che sia di formato numerico e che possa contenere una virgola o un segno meno utilizzando il seguente codice

```
Private Sub Text1_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
If InStr("1234567890,-", Chr(KeyAscii)) = 0 Then KeyAscii = 0
End Sub
```

Analizziamo questa riga: If InStr("1234567890,-", Chr(KeyAscii)) = 0 Then KeyAscii = 0

a. La funzione **Chr** restituisce una stringa contenente il carattere associato al codice carattere specificato nella variabile KeyAscii che corrisponde al codice del tasto premuto nella nostra procedura.
b. La funzione **InStr** (stringa1, stringa2), restituisce un valore di tipo Variant, che indica la posizione della prima occorrenza di una stringa (stringa2) all'interno di un'altra stringa (stringa1), nel nostro codice abbiamo: stringa1 = "1234567890, -" e stringa2 = Chr (KeyAscii)).
c. La funzione = 0 rappresenta un avviso che non è stata trovata l'occorrenza nella stringa di riferimento e quindi restituisce 0.
d. L'enunciato KeyAscii = 0 è un valore nullo assegnato al pulsante premuto che equivale ad annullare il testo digitato.

Molto brevemente con il codice sopra esposto verifichiamo se non c'è un'occorrenza del carattere corrispondente al tasto premuto nella nostra stringa di riferimento, in tal caso non prendiamo in considerazione il testo digitato. Testando il codice nella UserForm otteniamo l'effetto desiderato, ma l'utente potrebbe avere la sensazione che la sua tastiera possa avere un problema (non scrive i caratteri), si dovrebbe rimandare un messaggio di errore, ma pesante a quanti avvisi potrebbe ricevere l'utente utilizzando un metodo come questo.

Risulta più conveniente mettere un "**Beep**" tramite l'altoparlante di sistema per comunicare all'utente che il testo digitato non è permesso, pertanto il codice diventa:

```
Private Sub Text1_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
If InStr("1234567890,-", Chr(KeyAscii)) = 0 Then KeyAscii = 0 : Beep
End Sub
```

A questo punto è possibile modificare il codice per accettare un segno – (meno) che sarà posto all'inizio della stringa con l'aggiunta di un operatore (Or) nella seguente forma:

```
Private Sub Text1_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
If InStr("1234567890,-", Chr(KeyAscii)) = 0 Or Text1.SelStart > 0 And Chr(KeyAscii) = "-"  _
Then KeyAscii = 0 : Beep
End If
```

Analizziamo questa parte: Or Text1.SelStart> 0 And Chr (KeyAscii) = " - "

a. La proprietà **SelStart** indica l'inizio del testo selezionato, o il punto di inserimento e se non viene selezionato nessun testo, l'intervallo di valori validi è compreso tra 0 e il numero totale di caratteri nell'area di modifica del controllo e pertanto restituisce la posizione che è occupata dal carattere premuto, se questa non corrisponde al primo (0), l'inserimento viene cancellato
b. And Chr (KeyAscii) = "-" assicura che SelStart venga applicata solo a un carattere.

A questo punto, è possibile consentire di poter digitare una virgola in qualsiasi punto tra due numeri, ma una sola volta. Fondamentalmente abbiamo bisogno di un nuovo operatore (And) nella seguente forma:

```
Private Sub Text1_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
If InStr("1234567890,-", Chr(KeyAscii)) = 0 Or Text1.SelStart > 0 And Chr(KeyAscii) = "-" _
Or InStr(Text1.Value, ",") <> 0 And Chr(KeyAscii) = ","  Then KeyAscii = 0 : Beep
End If
```

Analizziamo questa parte: Or InStr (Text1.Value, ",") <> 0 And Chr (KeyAscii) = ","

a. Viene utilizzato, nuovamente, per verificare se *InStr* ha già un punto esistente nella stringa.
b. Se il carattere è un punto (che sarebbe troppo), viene rifiutato.

In sostanza, se c'è già un punto nella stringa, e se il carattere digitato è un punto, vale a dire che stiamo cercando di inserire un secondo punto, viene rifiutato. A questo punto si dovrebbe cercare di vietare il copia e incolla, compreso il controllo della clipboard, ma questo aspetto è più difficile in quanto non possiamo semplicemente prendere in considerazione solo il primo livello degli Appunti di Excel, ma dobbiamo anche considerare Windows, e questo è oltre la portata di questo tutorial, pertanto per il momento riteniamo che non c'è nessun evento copia/Incolla diretto al TextBox, quindi proseguiamo con l'analisi degli altri vincoli da rispettare che sono i seguenti:

1. Verificare che la stringa non è vuota.
2. Che non ha punti.
3. Se ha un solo carattere, e che non è parte di una delle 10 cifre, viene rifiutato.
4. Che si tratta di una stringa che rappresenta un numero come definito inizialmente.

Quindi useremo l'evento *Private Sub Text1_BeforeUpdate (ByVal Cancel As MSForms.ReturnBoolean)*, che si verifica prima di modificare i dati in un controllo, per controllare il valore della casella di testo prima che venga convalidato.

Poco sopra abbiamo detto che se il codice venisse utilizzato di frequente sarebbe indicato utilizzare, o meglio, creare una funzione personale sia per fornire ergonomia professionale alla futura applicazione, che per comodità. Quindi possiamo sfruttare l'opportunità per costruire una funzione a questo livello con il seguente codice:

```
Private Sub Text1_BeforeUpdate(ByVal Cancel As MSForms.ReturnBoolean)
Dim stringa1 As String
Stringa1 = Text1.Value
If FunOk(stringa1) = True Then Cancel = True: Text1.Value = "": Beep: MsgBox _
"Input non valido !"
End Sub
```

La nostra Funzione è stata chiamata "*FunOK*" e restituisce un valore booleano (True o False) che ci dirà se la stringa proposta dovrebbe essere respinta o no. Per analizzare la funzione useremo la variabile stringa1 a cui viene assegnato il valore della TextBox. La funzione "FunOK", presenta di default il valore False, mentre se il valore è True, la stringa viene scartata.

```
FunOK If (stringa1) = True Then Cancel = True: Text1.Value = " "Beep: MsgBox" Input non valido "
```

Che sta a indicare: Se il valore di ritorno è True, quindi il parametro Annulla della macro è vero, vuol dire che hai inserito una stringa vuota nel controllo TextBox, emetti un segnale acustico e visualizza il messaggio "Input non valido !". Per verificare se la stringa è vuota non c'è bisogno di avviare un processo, per questo, utilizziamo una nuova istruzione IF Then Else auto-esplicativa come la seguente:

```
If stringa1 = "" Then Exit Function
```

Mentre invece se non include nessun segno possiamo usare un'altra istruzione If Then Else nella forma:

```
If Len(Replace(stringa1, ".", "")) <> Len(stringa1) Then FunOK = True: Exit Function
```

Da ricordare che la funzione **Len** restituisce un valore long che contiene il numero di caratteri di una stringa, pertanto se la lunghezza della stringa ottenuta sostituendo il punto con una stringa vuota è diversa dalla lunghezza della stringa di base, significa che la nostra stringa ha un punto e quindi viene assegnato il valore di riferimento come True alla nostra funzione e il valore sarà respinto. Se invece la stringa ha un solo carattere, che non è parte di una delle 10 cifre permesse, allora viene rifiutata in questo modo:

```
If Len(stringa1) = 1 And InStr("1234567890", stringa1) = 0 Then FunOK = True: Exit Function
```

Il codice sopra esposto sta a indicare che: Se la lunghezza della stringa è uguale a 1 e non troviamo nessuna corrispondenza con le 10 cifre permesse, allora si assegna il valore True come valore di ritorno alla funzione e il valore verrà respinto. Dopo le prime operazioni eseguite, resta ora quello di assicurarsi che la stringa corrisponda a un numero secondo i nostri desideri (numeri da 0 a 9, un possibile punto positivo o negativo). Per fare questo, sarebbe conveniente utilizzare la funzione **Val** che restituisce il numero contenuto in una stringa come un valore numerico di tipo appropriato e se non viene trovato un numero restituisce 0 e non un messaggio di errore.

A sfavore di questa funzione c'è che le virgole non sono riconosciute, infatti l'editor di VBA riconosce solo il punto come separatore numerico. Quindi dovremo trattare la stringa per testare la sostituzione (solo per le prove) della virgola, potenzialmente esistente, con il punto che ci consente di utilizzare la funzione Val con il seguente codice:

```
stringa1 = Replace(stringa1, ",", ".")
```

A questo livello, ci si baserà su questo ulteriore caratteristica della funzione Val, in cui la funzione interrompe la lettura della stringa in corrispondenza del primo carattere che non è una parte evidente di un numero. E' questo quello che ci interessa, infatti, se si tratta di una sequenza eterogenea di caratteri che comprende lettere nel mezzo di numeri, la lettura della funzione *Val* si ferma al primo carattere non permesso incontrato. Allora possiamo costruire, come sopra, il metodo per confrontare la lunghezza della stringa risultante con l'originale e vedere, ovviamente, la differenza.

Si deve ricordare che la funzione *Len* si occupa solo di lunghezze di stringhe, poi attraverso l'utilizzo della funzione Val viene convertita in un'espressione numerica, e in seguito con la funzione **CStr** si converte un'espressione numerica in una stringa. Possiamo utilizzare un codice come il seguente:

```
If Len(CStr(Val(stringa1))) <> Len(stringa1) Then FunOK = True
```

Che consiste in: Se la lunghezza della stringa costituita dal primo carattere numerico incontrato, prima di un carattere nella stringa da testare, è diversa dalla lunghezza di quest'ultimo, vuol dire che c'è un carattere indesiderato, quindi influisce sul valore restituito, cioè True alla funzione e l'inserimento verrà respinto. A questo punto resta da scrivere il valore nell'apposita cella del foglio di lavoro che possiamo farlo usando il seguente codice:

```
Private Sub command1_Click()
Cells(5, 1) = Text1.Value
End Sub
```

E otteniamo:

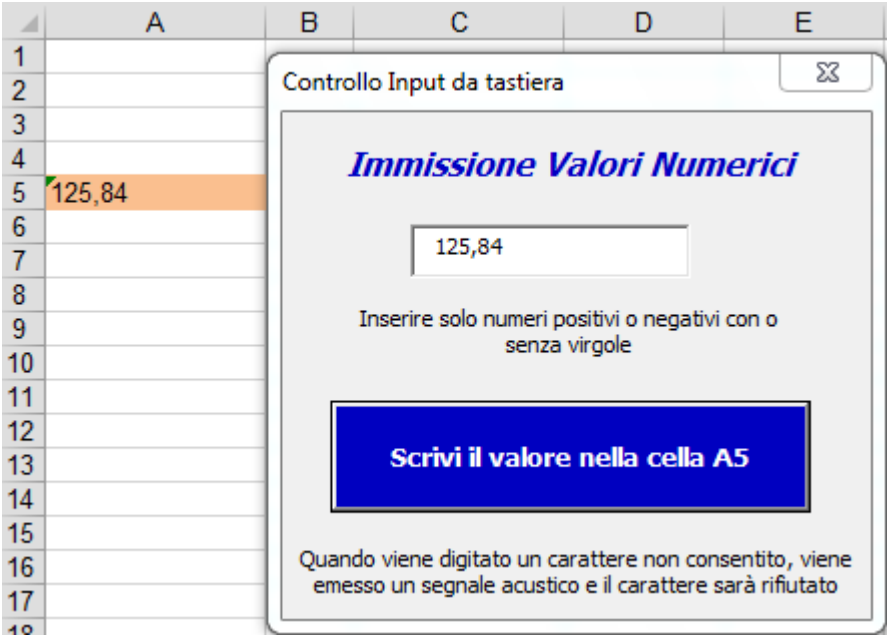


Fig. 1

L'allineamento predefinito a sinistra del nostro valore nella casella A5 indica che i dati scritti sono stati considerati come una stringa, come conferma l'attivazione del pop-up:

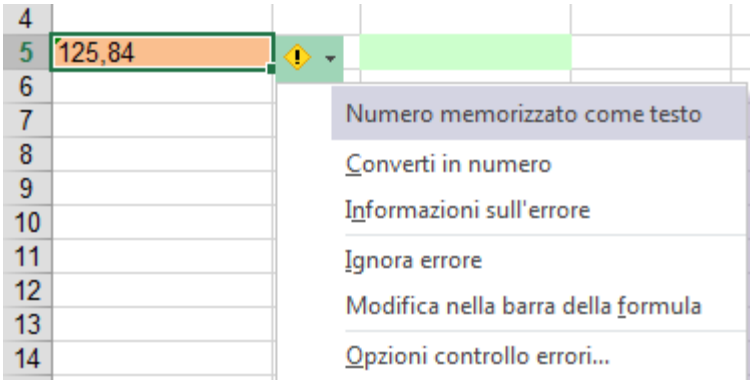


Fig. 2

Si prega di notare questo comportamento, che è la causa di molti malintesi, perché se includiamo quella cella in un calcolo come "*= A5 + A7*" con un numero in A7, avremo un risultato, mentre se si utilizza la funzione SOMMA "*= Somma (A5 : A7)*", il valore di A5 non sarà preso in considerazione, perchè, ancora una volta, è normale e segue le regole di interpretazione di Excel. Finora abbiamo visto un TextBox che restituisce valori di stringa, ma possiamo convertire il valore nel formato numerico oppure secondo i criteri della cella di destinazione usando il seguente codice:

```
Private Sub scrivi1_Click()
Cells(5, 3) = CDbl(Text1.Value)
End Sub
```

La funzione **CDbl** converte un'espressione (che può essere qualsiasi espressione stringa o un'espressione numerica) in una variabile di tipo Double, possiamo verificare scrivendo il valore nella cella C5 (verde) per visualizzare il risultato.

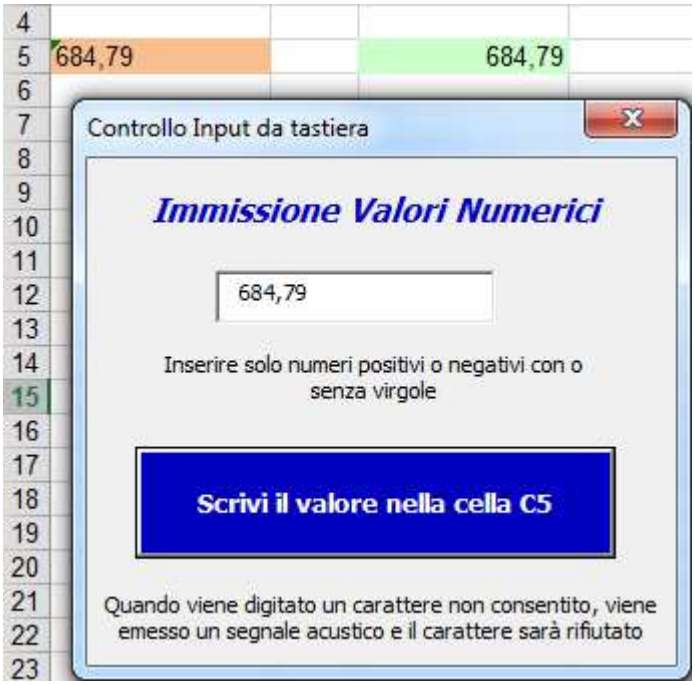


Fig. 3

Possiamo completare il codice per raggiungere i due tipi di dati da scrivere nella cella e vederne i risultati in questo modo:

```
Private Sub scrivi1_Click()
   MsgBox "Questo tipo di dati è :  " & TypeName(Text1.Value) & vbCr & vbLf _
   & "e abbiamo il testo nella cella A5 con allineamento predefinito a sinistra."
   Worksheets("Foglio1").Cells(5, 1) = Text1.Value

   Worksheets("Foglio1").Cells(5, 3) = CDbl(Text1.Value)
   MsgBox "Questo tipo di dati è :  " & TypeName(CDbl(Text1.Value)) & vbCr & vbLf _
   & "e abbiamo un numero in virgola mobile nella cella C5 con allineamento predefinito a destra"
End Sub
```

Il fatto di avere le colonne A e C estese, mette in evidenza le differenze di allineamento a causa del formato, è molto meno visibile (e fuorviante) sotto le colonne di base in funzione del numero di cifre!. In questa ultima fase abbiamo impiegato la funzione **TypeName** per visualizzare nel messaggio informativo che restituisce se una stringa corrisponde al tipo di una variabile. A questo punto il listato finale è il seguente:

```
Option Explicit
Private Sub scrivi1_Click()
   MsgBox "Questo tipo di dati è :  " & TypeName(Text1.Value) & vbCr & vbLf _
   & "e abbiamo il testo nella cella A5 con allineamento predefinito a sinistra."
   Worksheets("Foglio1").Cells(5, 1) = Text1.Value

   Worksheets("Foglio1").Cells(5, 3) = CDbl(Text1.Value)
   MsgBox "Questo tipo di dati è :  " & TypeName(CDbl(Text1.Value)) & vbCr & vbLf _
   & "e abbiamo un numero in virgola mobile nella cella C5 con allineamento predefinito a destra"
End Sub

Private Sub Text1_BeforeUpdate(ByVal Cancel As MSForms.ReturnBoolean)
Dim stringa1 As String
   stringa1 = Text1.Value
   If FunOK(stringa1) = True Then Cancel = True: Text1.Value = "": Beep: MsgBox "Input non
Valido !"
End Sub

Private Function FunOK(stringa1 As String) As Boolean
   If stringa1 = "" Then Exit Function
   If Len(Replace(stringa1, ".", "")) <> Len(stringa1) Then FunOK = True: Exit Function
   If Len(stringa1) = 1 And InStr("1234567890", stringa1) = 0 Then FunOK = True: Exit Function
   stringa1 = Replace(stringa1, ",", ".")
   If Len(CStr(Val(stringa1))) <> Len(stringa1) Then FunOK = True
End Function

Private Sub Text1_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
   If InStr("1234567890,-", Chr(KeyAscii)) = 0 Or Text1.SelStart > 0 And Chr(KeyAscii) = "-" _
     Or InStr(Text1.Value, ",") <> 0 And Chr(KeyAscii) = "," Then
      KeyAscii = 0: Beep
   End If
End Sub
```

Gli eventi della cartella di lavoro o ThisWorkbook

Per funzionare appropriatamente, la maggior parte dei programmi di Excel, VBA modifica in qualche modo l'ambiente dell'applicazione stessa, queste modifiche possono comportare l'aggiunta di comandi di menu, la visualizzazione di barre degli strumenti o l'impostazione di opzioni. Nella maggior parte dei casi, tutti questi cambiamenti devono essere apportati in Excel prima che l'utente avvii il programma di VBA, in modo che i comandi di menu e le barre degli strumenti siano disponibili fin dall'inizio.

Un programma di VBA non dovrebbe obbligare l'utente ad aggiungere menu e barre degli strumenti e a modificare le impostazioni delle opzioni, pertanto dobbiamo fare in modo che questi cambiamenti si verifichino automaticamente all'avvio usando una routine di gestione dell'evento **Open** della cartella di lavoro. L'evento Open viene generato quando viene aperta una cartella di lavoro e il codice di una routine Open, viene eseguito ogni volta che la cartella di lavoro viene aperta e rappresenta uno strumento ideale per impostare le eventuali condizioni speciali necessarie per la cartella stessa. Per poter accedere agli eventi della cartella si deve agire in questo modo:

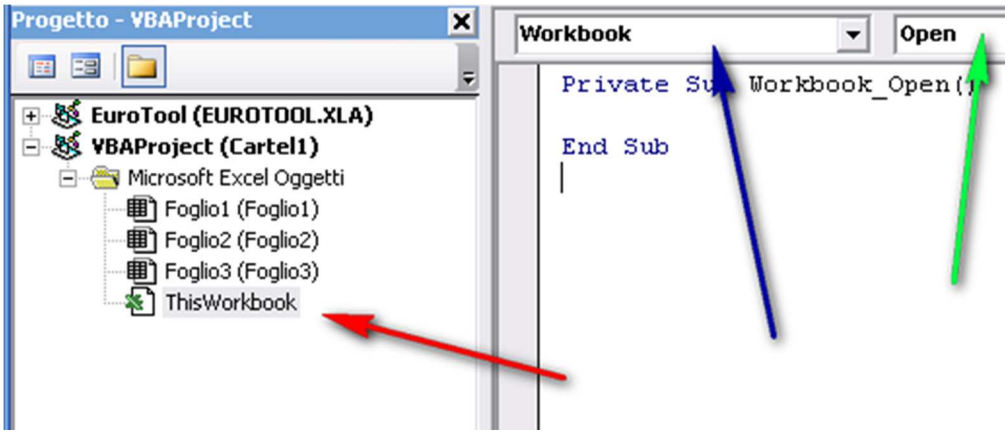


Fig. 1

Selezionare la voce *ThisWorkbook* nella finestra del progetto, indicato dalla *freccia rossa* in *figura 2*, per accedere al modulo di classe della cartella di lavoro (Workbook) di seguito selezionare Workbook nella casella a discesa sopra la finestra del codice, indicata dalla *freccia blu* e nella casella a fianco, indicata dalla *freccia verde* compariranno i vari eventi per l'oggetto. Di seguito riportiamo gli eventi principali associati all'oggetto WorkBook:

Private Sub Workbook_Open ()

Questo evento può innescare una procedura quando si apre la cartella di lavoro

```
Private Sub Workbook_Open()
    MsgBox "Buongiorno " & Environ("UserName")
End Sub

Private Sub Workbook_Open()
MsgBox "Benvenuto"
End Sub
```

*Si consiglia di impostare il livello di protezione macro sul livello **medio**, in modo che le macro non vengano attivate senza l'autorizzazione dell'utente, inoltre quando si apre una cartella di lavoro proveniente da una persona che non si conosce, è meglio aprire il file disattivando le macro per verificare che non contengano procedure che svolgono azioni indesiderate.*

Private Sub Workbook_Activate()

Questo evento viene innescato quando viene attivata la cartella di lavoro. La procedura non viene avviata se si proviene da un'altra applicazione, mentre la cartella di lavoro era attiva

```
Private Sub Workbook_Activate()
'nascondere la barra della formula
   Application.DisplayFormulaBar = False
End Sub

Private Sub Workbook_Activate()
 'mostra la userform1
Userform1.Show
End Sub
```

Private Sub Workbook_Deactivate ()

Questo evento viene attivato quando si seleziona un'altra cartella di lavoro.

```
Private Sub Workbook_Deactivate()
 'attiva la barra della formula
    Application.DisplayFormulaBar = True
```

Private Sub Workbook_BeforeClose (Cancel As Boolean)

Questo evento viene generato prima di chiudere la cartella di lavoro, che si chiude solo quando l'evento è terminato. Il parametro Cancel impedisce la chiusura del file.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
 'blocca la chiusura della cartella di lavoro finchè la cella A1 è vuota.
   If Sheets("Foglio1").Range("A1") = "" Then
       MsgBox "Completare l'inserimento in A1"
       Cancel = True
   Else
       ThisWorkbook.Save
   End If
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   If MsgBox("Vuoi veramente chiudere la cartella di lavoro?", 36, "Conferma") =
vbNo Then
       Cancel = True
   End If
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
 'salvare data e ora nel foglio1
     Worksheets("Foglio1").Range("A1").Value = _
        Format(Date + Time, "dd/mm/yy hh:mm")
     ThisWorkbook.Save
End Sub
```

Private Sub Workbook_BeforePrint (Cancel As Boolean)
Questo evento si verifica prima della stampa che inizia solo dopo questa procedura, il parametro Cancel blocca l'esecuzione della macro

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
 'ricalcola tutti i fogli della cartella di lavoro attiva prima di stampare
 For Each wk in Worksheets
 wk.Calculate
 Next
End Sub

Private Sub Workbook_BeforePrint(Cancel As Boolean)
  ActiveSheet.PageSetup.PrintArea = Selection
End Sub

Private Sub Workbook_BeforePrint(Cancel As Boolean)
Cancel = false
Worksheets("Foglio1").PageSetup.RightFooter = "Produced by " & myname
End Sub
```

Private Sub Workbook_BeforeSave (ByVal SaveAsUI As Boolean, Cancel As Boolean)
Questo evento viene generato prima di avviare il lavoro di backup, il parametro SaveAsUI restituisce TRUE se la casella "Salva con nome" verrà visualizzata, se si specifica il parametro Cancel = True verrà bloccata l'operazione di salvataggio

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As
Boolean)
 'impedire il salvataggio
    Cancel = True
End Sub

Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel as
Boolean)
    a = MsgBox("Vuoi salvare la cartella corrente?", vbYesNo)
    If a = vbNo Then Cancel = True
End Sub

Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As
Boolean)
    MsgBox "Arrivederci cartella salvata con successo"
End Sub
```

Private Sub Workbook_NewSheet (ByVal Sh As Object)
Questo evento viene attivato quando un nuovo foglio viene inserito nella cartella di lavoro, il parametro **Sh** è il foglio di lavoro creato

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
 'visualizza il nome e l'indice del nuovo foglio.
    MsgBox Sh.Name & " : " & Sh.Index
 End Sub

Private Sub Workbook_NewSheet(ByVal Sh as Object)
 'sposta nuovi fogli alla fine del lavoro
 Sh.Move After:= Sheets(Sheets.Count)
End Sub

Private Sub Workbook_NewSheet(ByVal Sh As Object)
Sheet1.UsedRange.Copy Sh.UsedRange
End Sub
```

Private Sub Workbook_WindowActivate (ByVal Wn As Window)
Questo evento si verifica quando si attiva la finestra che contiene la cartella di lavoro e il parametro Wn corrisponde alla finestra attiva

```
Private Sub Workbook_WindowActivate(ByVal Wn As Excel.Window)
 Wn.WindowState = xlMaximized
End Sub

Private Sub Workbook_WindowActivate(ByVal Wn As Window)
    MsgBox "Attivato"
End Sub

Private Sub Workbook_WindowActivate(ByVal Wn As Window)
Msgbox "Questa cartella è attiva"
End Sub
```

Private Sub Workbook_WindowDeactivate (ByVal Wn As Window)
Questo evento si verifica quando si disattiva la finestra che contiene la cartella di lavoro per attivare un'altra finestra di Excel e il parametro Wn corrisponde alla finestra disabilitata.

```
Private Sub Workbook_WindowDeactivate(ByVal Wn As Window)
    MsgBox "Disattivato"
End Sub

Private Sub Workbook_WindowDeactivate(ByVal Wn As Window)
 'nasconde la barra multifunzione se l'altezza è inferiore a 100
altezz = Application.CommandBars("Ribbon").Height
If altezz < 100 Then Application.SendKeys ("^{F1}")
End Sub

Private Sub Workbook_WindowDeactivate(ByVal Wn As Window)
MsgBox "Hai disattivato " & Wn.Caption
End Sub
```

Private Sub Workbook_WindowResize (ByVal Wn As Window)
Questo evento si verifica quando si ridimensiona la finestra che contiene la cartella di lavoro e il parametro **Wn** corrisponde alla finestra.

```
Private Sub Workbook_WindowResize(ByVal Wn As Excel.Window)
 'Inserisce nella barra di stato il nome del file e la scritta "ridimensionata"
 Application.StatusBar = Wn.Caption & " Ridimensionata"
End Sub

Private Sub Workbook_WindowResize(ByVal Wn As Window)
 'ridimensionando la finestra inserisce il valore in B1
    With ThisWorkbook.ActiveSheet.Range("B1")
        .Value = .Value + 1
    End With
End Sub

Private Sub Workbook_WindowResize(ByVal Wn As Window)
 'parcheggia la finestra se viene ridimensionata
Wn.WindowState = xlMinimized
End Sub
```

Private Sub Workbook_AddinInstall ()
Questo evento viene utilizzato nei componenti aggiuntivi (XLA) e viene attivato durante l'installazione del componente aggiuntivo contenuto nella procedura evento.

```
Private Sub Workbook_AddinInstall()
 'aggiunge un controllo alla barra degli strumenti standard
With Application.Commandbars("Standard").Controls.Add
 .Caption = "Aggiungi Voci al menu"
 .OnAction = "'pippo11.xls'!Amacro"
 End With End Sub
End Sub

Private Sub Workbook_AddinInstall()
Dim MyForm As Object
Set MyForm =
ActiveWorkbook.VBProject.VBComponents.Add(vbext_ct_MSForm)
    'nuova userform
    With MyForm
        .Properties("Caption") = "My Form"
        .Properties("Width") = 450
        .Properties("Height") = 300
    End With
End Sub

Private Sub Workbook_AddinInstall()
  Run "AddMenus"
End Sub
```

Private Sub Workbook_AddinUninstall ()
Questo evento viene utilizzato nei Componenti aggiuntivi e viene attivata quando il componente aggiuntivo viene disinstallato.

```
Private Sub Workbook_AddinUninstall()
  Run "DeleteMenu"
End Sub

Private Sub Workbook_AddinUninstall()
   Application.WindowState = xlMinimized
End Sub

Private Sub Workbook_AddinUninstall()
   For Each Component In ActiveWorkbook.VBProject.VBComponents
      If Component.Type = 3 Then
         ActiveWorkbook.VBProject.VBComponents.Remove Component
      End If
   Next Component
End Sub
```

Private Sub Workbook_BeforeXmlExport (ByVal Map As XmlMap, ByVal Url As String, Cancel As Boolean)
Questo evento si attiva quando Excel salva o esporta i dati XML da questa cartella di lavoro specifica, il parametro Map rappresenta la tabella xml di mappatura utilizzata per l'esportazione e il parametro Url restituisce il percorso completo del file xml che verrà utilizzato per l'esportazione. Si specifica Cancel = True per impedire l'esportazione.

```
Private Sub Workbook_BeforeXmlExport(ByVal Map As XmlMap, ByVal Url As
String, _
    Cancel As Boolean)
    MsgBox Map.Name & vbCrLf & Url
End Sub

Private Sub Workbook_BeforeXmlExport(ByVal Map As XmlMap, ByVal Url As
String, _
    Cancel As Boolean)
If (Map.IsExportable) Then
    If MsgBox("Microsoft Excel è pronto" & " per esportare in XML da " &
"Map.Name" & "." & vbCrLf & _
" Vuoi continuare?", vbYesNo + vbQuestion, "Processo di esportazione in XML")
= 7 Then Cancel = True
End If
End Sub

Private Sub Workbook_BeforeXmlExport(ByVal Map As XmlMap, _ ByVal Url As
String, Cancel As Boolean)
Debug.Print "Verifica prima di esportare", Map, Url, IsRefresh, Cancel
End Sub
```

Private Sub Workbook_AfterXmlExport (ByVal Map As XmlMap, ByVal Url As String, ByVal Result As XlXmlExportResult)
Questo evento viene attivato dopo l'esportazione dei dati in un file xml (da Excel2003), il parametro Map rappresenta la tabella xml di mappatura utilizzata per l'esportazione il parametro URL restituisce il percorso completo del file xml salvato durante l'esportazione, mentre invece il parametro Result restituisce il risultato dell'esportazione che può essere:

- *0 (xlXmlExportSuccess)*: L'esportazione è stata eseguita correttamente oppure
- *1 (xlXmlExportValidationFailed)*: L esportazione non riuscita

```
Private Sub Workbook_AfterXmlExport(ByVal Map As XmlMap, ByVal Url As
String, ByVal Result As XlXmlExportResult)
   MsgBox Map.Name & vbCrLf & _
      Url & vbCrLf & _
      Result
End Sub
```

Private Sub Workbook_BeforeXmlImport (ByVal Map As XmlMap, ByVal Url As String, ByVal IsRefresh As Boolean, Cancel As Boolean)
Questo evento si attiva quando si importano dei dati da un file XML nel foglio di lavoro. Il parametro Map rappresenta i dati di mapping che importano il file XML e il parametro URL restituisce il percorso completo del file XML importato. Il parametro *IsRefresh* identifica se l'importazione è di una nuova origine dati o se è un aggiornamento esistente nel foglio. Se viene restituito True è un aggiornamento, si specifica Cancel = True per impedire l'importazione.

```
Private Sub Workbook_BeforeXmlImport(ByVal Map As XmlMap, ByVal Url As
String, ByVal IsRefresh As Boolean, Cancel As Boolean)
MsgBox Map.Name & vbCrLf & Url
End Sub

Private Sub Workbook_BeforeXmlImport(ByVal Map As XmlMap, ByVal Url As
String, ByVal IsRefresh As Boolean, Cancel As Boolean)
   MsgBox IsRefresh & vbCrLf & _
      Map.Name & vbCrLf & _
      Url
End Sub

Private Sub Workbook_BeforeXmlImport(ByVal Map As XmlMap, ByVal Url As
String, ByVal IsRefresh As Boolean, Cancel As Boolean)
Debug.Print "Verifica prima di importare", Map, Url, IsRefresh, Cancel
End Sub

Private Sub Workbook_BeforeXmlImport(ByVal Map As XmlMap, ByVal Url As
String, ByVal IsRefresh As Boolean, Cancel As Boolean)
If Map.RootElementName = "Modello_pippo" Then
  MsgBox "Il file XML che hai selezionato non può essere importato"
Cancel = True
End If
End Sub
```

Private Sub Workbook_AfterXmlImport (ByVal Map As XmlMap, ByVal IsRefresh As boolean, ByVal result As XlXmlImportResult)
Questo evento viene attivato dopo l'inserimento o l'aggiornamento dei dati XML nel foglio di lavoro e il parametro Map rappresenta la tabella xml di mappatura derivante dall'importazione e il parametro IsRefresh identifica se l'importazione è di una nuova origine dati o se si sta aggiornando un foglio di lavoro esistente, e viene restituito True se si tratta di un aggiornamento. Il parametro result restituisce il risultato dell'importazione, che può essere:

- *0 (xlXmlImportSuccess)*: L'importazione è riuscita
- *1 (xlXmlImportElementsTruncated)*: Il contenuto del file xml è stato troncato durante l'importazione perché è troppo grande per il foglio di lavoro.
- *2 (xlXmlImportValidationFailed)*: L'importazione non è riuscita

```
Private Sub Workbook_AfterXmlImport(ByVal Map As XmlMap, ByVal IsRefresh
As Boolean, _
    ByVal Result As XlXmlImportResult)
    MsgBox IsRefresh & vbCrLf & _
        Map.Name & vbCrLf & _
        Result
End Sub
```

Private Sub Workbook_PivotTableOpenConnection (ByVal Target As pivot)
L'evento è attivato quando una tabella pivot si connette a un'origine dati e il parametro Target è corrisponde alla tabella

```
Private Sub Workbook_PivotTableOpenConnection(ByVal Target As PivotTable)
    MsgBox "La connessione alla tabella Pivot è stata aperta"
End Sub

Private Sub Workbook_PivotTableOpenConnection(ByVal Target As PivotTable)
Application.DisplayAlerts = False
End Sub
```

Private Sub Workbook_PivotTableCloseConnection (ByVal Target As pivot)
L'evento è attivato quando una tabella pivot è scollegata dalla sorgente dati e il parametro Target corrisponde alla tabella offline.

Private Sub Workbook_RowsetComplete (ByVal Description As String, ByVal Sheet As String, ByVal Success As Boolean)
Si verifica quando l'utente esamina il recordset o chiama il set di righe per un oggetto tabella pivot (da Excel2007). Il parametro *Description* restituisce una breve descrizione dell'evento e il parametro Sheet restituisce il nome del foglio che contiene il Recordset creato.

Private Sub Workbook_Sync (ByVal SyncEventType As Office.MsoSyncEventType)
Viene generato quando la copia in locale di un foglio di lavoro che fa parte del lavoro di un documento è sincronizzato con la copia sul server (da Excel2003) Il parametro SyncEventType può assumere i seguenti valori:

- *msoSyncEventDownloadFailed*
- *msoSyncEventDownloadInitiated*
- *msoSyncEventDownloadNoChange*
- *msoSyncEventDownloadSucceeded*
- *msoSyncEventOffline*
- *msoSyncEventUploadFailed*
- *msoSyncEventUploadInitiated*
- *msoSyncEventUploadSucceeded*

```
Private Sub Workbook_Sync(ByVal SyncEventType As Office.MsoSyncEventType)
   If SyncEventType = msoSyncEventDownloadFailed Or _
         SyncEventType = msoSyncEventUploadFailed Then _
         MsgBox "La sincronizzazione è fallita"
End Sub
```

*Tutti gli eventi il cui nome inizia con **Workbook_Sheet** hanno il loro equivalente in ogni modulo foglio e il principio di funzionamento è identico. Si utilizza la procedura nel modulo **ThisWorkbook** quando si necessita di scrivere un unico processo per la gestione di tutti i fogli della cartella di lavoro.*

Private Sub Workbook_SheetActivate (ByVal Sh As Object)
Identifica l'attivazione di un foglio nella cartella di lavoro e il parametro Sh corrisponde al foglio selezionato

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
 'restituisce il nome del foglio selezionato.
   MsgBox Sh.Name
End Sub

Private Sub Workbook_SheetActivate(ByVal Sh As Object)
   MsgBox "Nome del Foglio : " & Sh.Name
End Sub

Private Sub Workbook_SheetActivate(ByVal Sh As Object)
   Sh.Calculate
End Sub
```

Private Sub Workbook_SheetDeactivate (ByVal Sh As Object)
Si verifica quando un foglio nella cartella di lavoro viene disattivato e il parametro Sh corrisponde al foglio disabilitato

```
Private Sub Workbook_SheetDeactivate(ByVal Sh As Object)
 MsgBox Sh.Name
End Sub

Private Sub Workbook_SheetDeactivate(ByVal Sh As Object)
MsgBox ("Foglio Disattivato")
End Sub

Private Sub Workbook_SheetDeactivate(ByVal Sh As Object)
 'parcheggiare la finestra di excel
   ThisWorkbook.Windows(1).WindowState = xlMinimized
End Sub
```

Private Sub Workbook_SheetChange (ByVal Sh As Object, ByVal Target As Range)
Questo evento viene attivato quando avvengono dei cambiamenti in una cella della cartella di lavoro, il parametro Sh corrisponde al foglio contenente la cella modificata e il parametro Target corrisponde alla cella modificata. Questo esempio identifica la cella che avete appena modificato.

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As
Range)
   'identifica la cella modificata
    If Target.Count > 1 Then Exit Sub
    MsgBox "Hai modificato la cella  " & Target.Address & _
        " (" & Target.Value & ")" & _
        " Nel foglio denominato " & Sh.Name
End Sub

Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As
Range)
 'incrementare la cella A1
    Application.EnableEvents = False
    Sheets(1).Range("A1") = Sheets(1).Range("A1") + 1
    Application.EnableEvents = True
End Sub

Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As
Range)
 'inserire il carattere grassetto e il colore rosso nella cella modificata
Target.Font.Bold = True
Target.Font.Color = vbRed
End Sub
```

Private Sub Workbook_SheetBeforeRightClick (ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)
Questo evento si verifica quando si utilizza il pulsante destro del mouse in uno dei fogli della cartella di lavoro e viene attivato dopo *WorkSheet_BeforeRightClick*. Il parametro Sh corrisponde al foglio attivo e il parametro Target è la cella che riceve il tasto destro del mouse, mentre il parametro Cancel disabilita l'azione associata della macro.

```
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target
As Range, Cancel As Boolean)
 'disabilita la visualizzazione del menu contestuale in tutti i fogli della cartella di
lavoro
    Dim x As Long, lngColorIndex As Long
    Cancel = True
        lngColorIndex = Application.Dialogs(xlDialogPatterns).Show
          x = ActiveCell.Interior.ColorIndex
        If lngColorIndex = xlColorIndexAutomatic Then x = xlColorIndexNone
    ActiveCell.Interior.ColorIndex = x
End Sub
```

Private Sub Workbook_SheetBeforeDoubleClick (ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)
Identifica il doppio click in una cella. Questo evento viene attivato dopo Worksheet_BeforeDoubleClick. Parametro Sh corrisponde alla scheda attiva e il parametro Target è la cella che riceve un doppio clic. il parametro Cancel disabilita l'azione della macro associata

```
Private Sub Workbook_SheetBeforeDoubleClick(ByVal Sh As Object, ByVal
Target As Range, Cancel AsBoolean)
    If Sh.Name = "Foglio1" Then
        Target.Interior.Color = RGB(255, 108, 0) 'Arancione
    Else
        Target.Interior.Color = RGB(136, 255, 0) 'Verde
    End If
End Sub

Private Sub Workbook_SheetBeforeDoubleClick(ByVal Sh As Object, _
 ByVal Target As Range, ByVal Cancel As Boolean)
 'disabilitare il doppio click
 Cancel = True
End Sub

Private Sub Workbook_SheetBeforeDoubleClick(ByVal Sh As Object, ByVal
Target As Range, Cancel As Boolean)
MsgBox "Hai fatto doppio click nel Foglio:  " & Sh.Name & vbLf & "e nella Cella:
" & Target.Address
End Sub
```

Private Sub Workbook_SheetCalculate (ByVal Sh As Object)
Questo evento viene attivato quando si esegue il ricalcolo (formule di convalida o di aggiornamento) in uno dei fogli della cartella di lavoro. L'evento si verifica dopo Worksheet_Calculate e il parametro Sh corrisponde al foglio che contiene la formula.

Private Sub Workbook_SheetFollowHyperlink (ByVal Sh As Object, ByVal Target As Hyperlink)
L'evento si verifica quando viene attivato un collegamento ipertestuale e il parametro Sh corrisponde al foglio che contiene il collegamento, mentre il parametro Target è il collegamento ipertestuale dell'oggetto

```
Private Sub Workbook_SheetFollowHyperlink(ByVal Sh As Object, ByVal Target
As Hyperlink)
 `mostra l'indirizzo del collegamento attivato
    MsgBox Target.Address & vbCrLf & Target.SubAddress
End Sub
```

Private Sub Workbook_SheetPivotTableUpdate (ByVal Sh As Object, ByVal Target As pivot)
Questo evento si verifica quando si aggiorna una tabella pivot, il parametro Sh corrisponde a foglio che contiene la tabella e il parametro Target è l'oggetto Tabella pivot aggiornato.

```
Private Sub Workbook_SheetPivotTableUpdate(ByVal Sh As Object, ByVal Target
As PivotTable)
    MsgBox "E' stata aggiornata " & Sh.Name & " / " & Target.Name
End Sub
```

Private Sub Workbook_SheetSelectionChange (ByVal Sh As Object, ByVal Target As Range)
Questo evento si verifica quando viene modificata la selezione di una cella in uno dei fogli della cartella di lavoro e viene attivato dopo Worksheet_SelectionChange. Il parametro Sh è il foglio attivo e il parametro Target corrisponde alla cella selezionata.

```
Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, ByVal Target
As Range)
 `recuperare il nome del foglio e l'indirizzo della cella
    MsgBox "Hai selezionato la cella  " & Target.Address & _
        " nel foglio denominato " & Sh.Name
End Sub
```

I Controlli CheckBox - OptionBox e ToggleButton

Il valore della proprietà di un *CheckBox* indica se è selezionato o meno. Un valore True indica che la casella è selezionata, False indica che è deselezionato e il valore *Null* indica che non è né attivo e né disattivo, e il CheckBox apparirà ombreggiato in questo caso. Per usare le 3 impostazioni si deve impostare Il valore della proprietà *TripleState* che può essere impostato sia nella finestra Proprietà o utilizzando una macro o codice VBA

Esempio: Visualizzare il valore del CheckBox, indicando se è selezionato o in uno stato Null

```
Private Sub UserForm_Initialize()
With Me.CheckBox1
.TextAlign = fmTextAlignCenter
.TripleState = True
End With
End Sub

Private Sub CheckBox1_Change()
If CheckBox1.Value = True Then
MsgBox "True"
ElseIf CheckBox1.Value = False Then
MsgBox "False"
Else
MsgBox "Null"
End If
End Sub
```

Esempio: Attivare una casella di testo TextBox solo se è selezionato il CheckBox

```
Private Sub UserForm_Initialize()
Me.TextBox1.Enabled = False
End Sub

Private Sub CheckBox1_Click()
If CheckBox1.Value = True Then
TextBox1.Enabled = True
Else
TextBox1.Enabled = False
End If
End Sub
```

Il Controllo OptionButton

Il controllo *OptionButton* è usato per fare una scelta tra più opzioni ed è indicato anche come Radio Button, che sceglie un'opzione da un gruppo di opzioni che si escludono a vicenda. Se sono presenti vari OptionButtons in una Form e non sono raggruppati, selezionandone uno si deseleziona tutte gli altri presenti nella Form. Tutti gli OptionsButtons all'interno di un gruppo specifico si escludono a vicenda all'interno di quel gruppo e non influenzano la selezione di altre OptionButtons di fuori di tale gruppo, inotre possono essere raggruppati con le seguenti modalità:

- Utilizzando la proprietà *GroupName* : Gli optionButtons possono essere prima aggiunti a un Form e poi assegnato ad un gruppo tramite la proprietà GroupName, in questo modo gli OptionButtons aventi lo stesso gruppo si escludono a vicenda selezionandone uno
- Utilizzando un controllo *Frame*: Prima si aggiungere il controllo Frame alla Form e quindi si disegnano gli OptionButtons all'interno della stessa. Ogni Frame separa gli OptionButtons presenti al suo interno da altri presenti in altri frame indipendentemente che gli altri abbiano lo stesso GroupName. Tuttavia, potrebbe essere preferibile raggruppare gli OptionButtons utilizzando la struttura GroupName invece di usare un frame solo per questo scopo, per risparmiare spazio e ridurre le dimensioni del form.
- Utilizzando un controllo *Multipage* che è anch'esso un contenitore e può essere utilizzato per raggruppare o organizzare i controlli all'interno di ciascuna delle sue pagine. Ogni pagina di un controllo multipage separa gli OptionButtons, indipendentemente dal fatto che tutte le OptionButtons (in tutte le pagine) hanno lo stesso GroupName .

Il valore di proprietà di un OptionButton indica se si è selezionato o meno. Un valore True indica che è selezionato, False è il valore di default e indica che non è selezionata.

Esempio: Determinare quali controlli all'interno di un Frame vengono selezionati

```
Private Sub CommandButton1_Click()
Dim ctrl As Control
For Each ctrl In Frame1.Controls
If ctrl.Value = True Then
MsgBox ctrl.Caption
End If
Next
End Sub
```

Esempio: Determinare quale OptionButton all'interno di un Frame, è selezionato:

```
Private Sub CommandButton1_Click ()
Dim ctrl As Control
For Each ctrl In Frame1.Controls
If TypeOf ctrl Is msforms.OptionButton Then
If ctrl.Value = True Then
MsgBox ctrl.Caption & " è selezionato", vbOKOnly, "Seleziona un OptionButton"
End If
End If
Next ctrl
End Sub
```

Esempio: Determinare quale CheckBoxes all'interno di un frame, vengono selezionati:

```
Private Sub CommandButton1_Click()
Dim ctrl As Control
For Each ctrl In Frame1.Controls
If TypeOf ctrl Is msforms.CheckBox Then
If Not TypeOf ctrl Is msforms.OptionButton Then
If ctrl.Value = True Then
MsgBox ctrl.Caption & " is selected", vbOKOnly, "Selected CheckBoxes"
End If
End If
End If
Next ctrl
End Sub
```

Si deve tener presente che il test per il CheckBoxe, cioè *TypeOf*, potrebbe non essere risolutivo, i controlli OptionButton e ToggleButton controllano anche l'interfaccia della casella e soddisfano questa condizione

Esempio: Determinare quale OptionButton, in una Form, con un particolare GroupName, viene selezionato.

```
Private Sub CommandButton1_Click()
Dim ctrl As Control
For Each ctrl In Me.Controls
If TypeOf ctrl Is msforms.OptionButton Then
If ctrl.GroupName = "GroupA" Then
If ctrl.Value = True Then
MsgBox ctrl.Caption & " è selezionato", vbOKOnly, "Seleziona un OptionButton"
End If
End If
End If
Next ctrl
End Sub
```

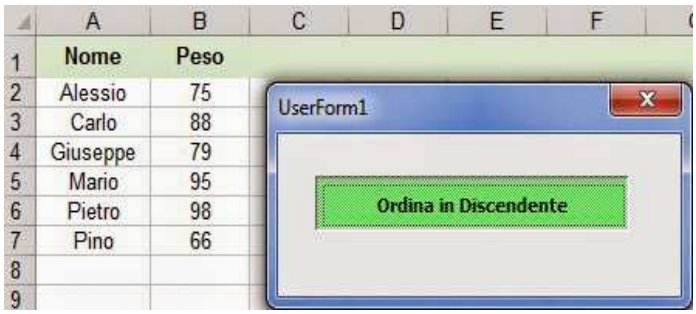
Con un *CheckBox*, l'utente può selezionare più opzioni, selezionando ogni CheckBox che rappresenta un'opzione disponibile nella scelta da eseguire, mentre invece un OptionButton viene utilizzato quando l'utente è autorizzato a selezionare solo una singola opzione. CheckBox può essere utilizzato in modo singolare, cioè un singolo CheckBox può essere selezionato o deselezionato (cliccando sul controllo) che indica se l'opzione deve essere esercitata o meno. OptionButton invece vengono utilizzati in multipli di o più, per indicare la scelta tra varie opzioni in modo esclusivo, inoltre viene deselezionato solo selezionando un altro OptionButton e non facendo clic su se stesso

Il controllo ToggleButton

Il controllo *ToggleButton* è un controllo che esegue un'azione quando si clicca una volta e il pulsante rimane premuto, e un'azione diversa quando si clicca una seconda volta e si rilascia il pulsante. In sostanza si comporta come un normale interruttore e dispone di due possibilità, On e Off. Si può avere un valore True quando appare come premuto o False quando appare non premuto. ToggleButton sembra un incrocio tra un CheckBox (funzionalità toggle) e un CommandButton (aspetto cliccabile e simili).

NB *ToggleButton può anche avere un valore Null (né selezionato né deselezionato) in cui apparirà ombreggiato, se la proprietà TripleState è impostata su true.*

Esempio: Utilizzare un controllo ToggleButton per alternare l'ordinamento tra ordine ascendente o ordine discendente di un intervallo di celle



Quando ToggleButton viene premuto, i dati nella colonna A e B vengono ordinati in ordine crescente e il pulsante viene visualizzato di colore verde, quando viene rilasciato le colonne vengono ordinate in ordine decrescente e il colore del pulsante diventa rosso

```
Private Sub UserForm_Initialize()
Dim totalrows As Long
totalrows = Foglio3.Cells(Rows.Count, "A").End(xlUp).Row
Me.ToggleButton1.Value = True
Me.ToggleButton1.Caption = "Ordina in Discendente"
Me.ToggleButton1.Font.Bold = True
Me.ToggleButton1.BackColor = RGB(0, 255, 0)
Foglio3.Range("A2:B" & totalrows).Sort Key1:=Foglio3.Range("A2"), Order1:=xlAscending
End Sub

Private Sub ToggleButton1_Click()
Dim totalrows As Long
totalrows = Foglio3.Cells(Rows.Count, "A").End(xlUp).Row
If Me.ToggleButton1.Value = True Then
Me.ToggleButton1.Caption = "Ordina in Discendente"
Me.ToggleButton1.BackColor = RGB(0, 255, 0)
Foglio3.Range("A2:B" & totalrows).Sort Key1:=Foglio3.Range("A2"), Order1:=xlAscending
ElseIf Me.ToggleButton1.Value = False Then
Me.ToggleButton1.Caption = "Ordina in Ascendente"
Me.ToggleButton1.BackColor = RGB(255, 0, 0)
Foglio3.Range("A2:B" & totalrows).Sort Key1:=Foglio3.Range("A2"), Order1:=xlDescending
End If
End Sub
```

Esempio: Utilizzare ToggleButton per nascondere o scoprire colonne o righe:

```
Private Sub ToggleButton1_Click()
If ToggleButton1.Value = True Then
Foglio3.Columns("B").EntireColumn.Hidden = True
Foglio3.Columns("C").EntireColumn.Hidden = False
Else
Foglio3.Columns("B").EntireColumn.Hidden = False
Foglio3.Columns("C").EntireColumn.Hidden = True
End If
End Sub

Private Sub ToggleButton1_Click()
Rows("1:3").Hidden = Not Rows("1:3").Hidden
End Sub
```

Esempio: Utilizzare CheckBox per nascondere o scoprire colonne o righe

```
Private Sub CheckBox1_Click()
If CheckBox1.Value = True Then
Foglio3.Columns("B").EntireColumn.Hidden = True
Foglio3.Columns("C").EntireColumn.Hidden = False
Else
Foglio3.Columns("B").EntireColumn.Hidden = False
Foglio3.Columns("C").EntireColumn.Hidden = True
End If
End Sub
```

I Controlli Frame, Multipage e TabStrip

I **frame** (cornici) vengono utilizzati per i controlli di gruppo che lavorano insieme e sono correlati tra loro o hanno qualche comunanza, in una Form, inoltre per raggruppare e organizzare una serie di elementi correlati, migliorano anche il layout della Form. Ad esempio, in un modulo che contenga le caratteristiche fisiche come l'altezza, peso e colore dei capelli possono essere raggruppati in un particolare frame. I Frame sono particolarmente utili per raggruppare due o più OptionButtons e vengono utilizzati per due finalità:

- Per raggruppare e organizzare in una Form i controlli e migliorare visivamente il layout
- Per impostare il comportamento di un gruppo di OptionButtons, che si escludono a vicenda all'interno di un frame e selezionando un OptionButton saranno deselezionate tutte le altre OptionButtons all'interno del Frame.

Esempio: Determinare il nome e la Caption di tutte le OptionButtons in un Frame

```
Private Sub CommandButton1_Click()
Dim ctrl As Control
For Each ctrl In Frame1.Controls
If TypeOf ctrl Is MSForms.OptionButton Then
If ctrl.Enabled = True Then
MsgBox ctrl.Name & " è abilitato con Caption " & ctrl.Caption
End If
End If
Next
End Sub
```

Esempio: Utilizzo di controlli in una cornice con codice VBA

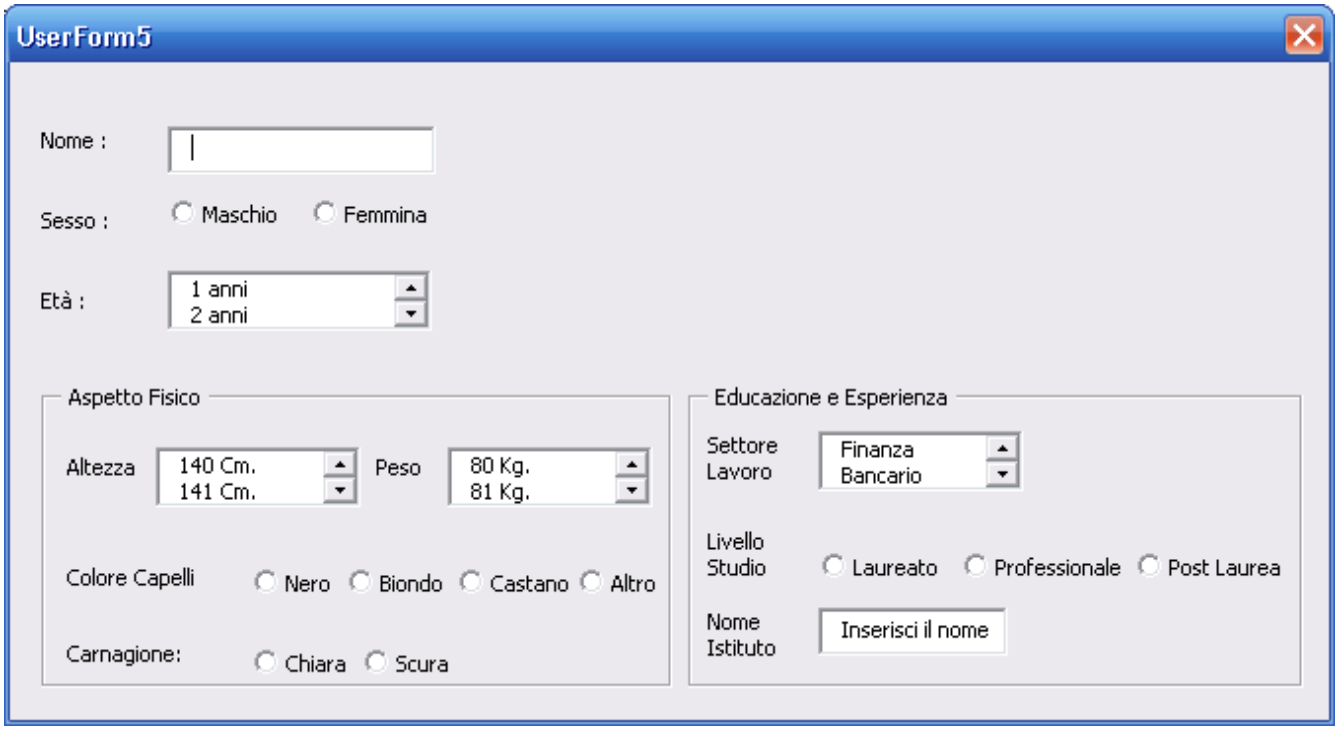


Fig. 1

```
Private Sub UserForm_Initialize()
   Dim i As Integer, myArray As Variant
Me.TextBox1.Value = "Inserisci il tuo Nome"
Me.Frame1.Caption = "Aspetto Fisico"
Me.Frame2.Caption = "Educazione e Esperienza"
With Me.ListBox1
For i = 1 To 100
.AddItem i & " anni"
Next i
End With

With Me.Frame1.ListBox2
For i = 140 To 200
.AddItem i & " Cm."
Next i
End With
With Me.Frame1.ListBox3
For i = 80 To 250
.AddItem i & " Kg."
Next i
End With

myArray = Array("Finanza", "Bancario", "Medicina", "Ingegneria", "Marketing",
"Management",  "Altro")
Me.Frame2.ListBox4.List = myArray
With Me.Frame2.ListBox4
For i = 1 To 50
.AddItem i & " anni"
Next i
End With

With Me.Frame1
OptionButton7.GroupName = "carnagione"
OptionButton8.GroupName = "carnagione"
OptionButton3.GroupName = "Capelli"
OptionButton4.GroupName = "Capelli"
OptionButton5.GroupName = "Capelli"
OptionButton6.GroupName = "Capelli"
End With

Me.Frame2.OptionButton9.GroupName = "studio"
Me.Frame2.OptionButton10.GroupName = "studio"
Me.Frame2.OptionButton11.GroupName = "studio"
Me.Frame2.TextBox2.Value = "Inserisci il nome"
End Sub

Private Sub TextBox1_Enter()
Me.TextBox1.Value = ""
End Sub

Private Sub TextBox2_Enter()
Me.Frame2.TextBox2.Value = ""
End Sub
```

Il controllo MultiPage

Un controllo *multipage* si compone di uno o più oggetti pagina, ciascuna contenente un proprio set di controlli e viene utilizzato al meglio quando si desidera gestire una grande quantità di dati che possono essere classificati in diverse categorie, in cui è possibile creare una pagina separata per ogni categoria. Tutti i controlli che vengono aggiunti a una pagina di un controllo multipage, sono contenuti diventano una parte di quella pagina che distingue i controlli da quelli che sono in altre parti della Form. Un controllo multipagina ha diverse pagine e la selezione di una pagina la rende quella corrente (rendendola visibile), nascondendo le altre e ogni pagina di un controllo multipage ha un proprio ordine di tabulazione. Le pagine sono numerate da 0, e per selezionare la prima pagina in un controllo multipagina, si utilizza il codice: *MultiPage1.Value = 0* e per impostazione predefinita, un controllo multipage ha 2 Pagine e per aggiungere delle pagine, si deve fare clic col destro del mouse sulla scheda e selezionare dal menu che appare la voce *Nuova pagina*

Aggiungi o Rimuovi pagina

Per aggiungere una pagina con codice VBA si utilizzo il *Metodo Add* che presenta questa sintassi:

Set m = MultiPage1.Pages.Add (pageName, pageCaption, pageIndex), dove:

- *pageIndex* (opzionale) è un intero che specifica la posizione della pagina da inserire, a partire da 0 per la prima posizione
- *pageName* imposta il nome per la pagina
- *pageCaption* imposta la Caption ed entrambe sono opzionali.

Alcuni esempi di inserimento di nuove pagine:

Set m = MultiPage1.Pages.Add ("Page5", "NewPage", 1)
Questo codice aggiunge una nuova pagina con il nome *Pagina5* e Caption *Nuova Pagina*, come seconda pagina, cioè nella seconda posizione nell'ordine di tabulazione.

MultiPage1.Pages.Add "Pagina3"
MultiPage1.Pages (2). Caption = "Nuova"
Questi 2 codici aggiungono una nuova pagina (la terza) con il nome *Pagina3* e imposta la Caption *Nuova*

MultiPage1.Pages.Add: Questo codice aggiunge semplicemente una nuova pagina.

Per rimuovere una pagina si usa il seguente metodo che presenta la sintassi:
MultiPage1.Pages.Remove (pageIndex)

Esempio: MultiPage1.Pages.Remove (1), questo codice rimuove la seconda pagina.

Individuare e Accedere a una pagina

Per modificare o impostare le proprietà di una pagina in fase di esecuzione, abbiamo bisogno di identificare quella pagina nel controllo multipage, che può essere fatto in diversi modi. Per accedere a una singola pagina in un controllo multipagina, possono essere usati i seguenti metodi

- *Numeric Index* : Con Index (indice) 0 ci si riferisce alla prima pagina, l'indice 1 alla seconda pagina e così via. Il codice per impostare la Caption è :*MultiPage1.Pages (Index).Caption*
- *Metodo Item*: L'indice (Item) 0 si riferisce alla prima pagina, l'indice 1 alla seconda pagina e così via. Il codice per impostare la Caption è :*MultiPage1.Pages.Item (ItemIndex).Caption*
- *Page Name*: Per impostare la Caption usando il nome pagina si usa il seguente codice: *MultiPage1.Pages ("Nuova Pagina"). Caption* oppure*MultiPage1.Pages.Item ("Nuova pagina"). Caption*
- *Page Object* : Il codice per impostare la Caption è *MultiPage1.PageName.Caption*
- *Proprietà SelectedItem* : Il codice per impostare la Caption è *MultiPage1.SelectedItem.Caption*

Esempio: Modificare le proprietà di ogni pagina di un controllo multipage, utilizzando diversi metodi di selezione pagina.

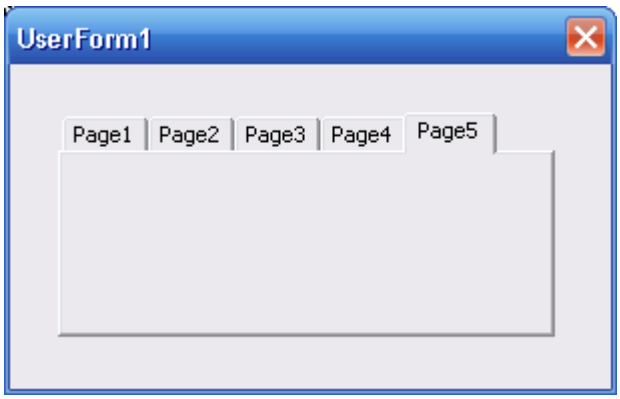


Fig. 2

Fig. 3

```
Private Sub UserForm_Activate()
MultiPage1.Pages(0).Caption = "Elisa"
MultiPage1.Pages.Item(1).Caption = "Daniele"
MultiPage1.Pages("Page3").Caption = "Mario"
MultiPage1.Page4.Caption = "Alice"
MultiPage1.Value = 4
MultiPage1.SelectedItem.Caption = "Genny"
End Sub
```

Creare una procedura guidata utilizzando un Form unico con controllo multipage

Nel caso in cui si desidera accettare dati in modo sequenziale (cioè step by step), allora invece di utilizzare più form è meglio utilizzare un controllo multipagina per creare più pagine in un form singolo. La pagina successiva può essere resa accessibile solo dopo che la pagina precedente è stata completata con tutti i dati. Mostriamo come creare un wizard composto da 4 pagine, come indicato di seguito.

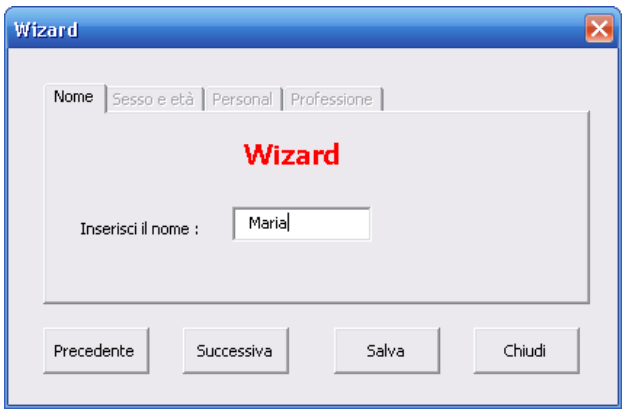


Fig. 4

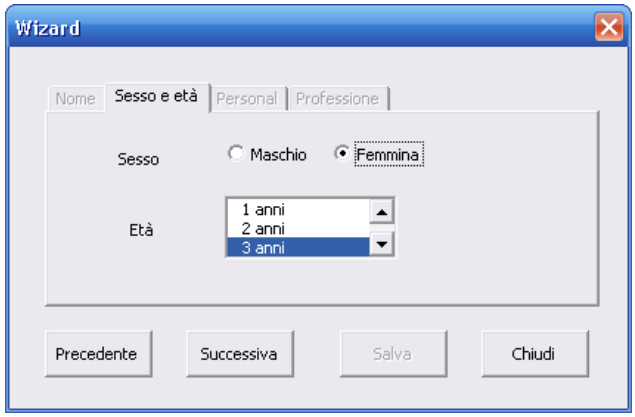


Fig. 5

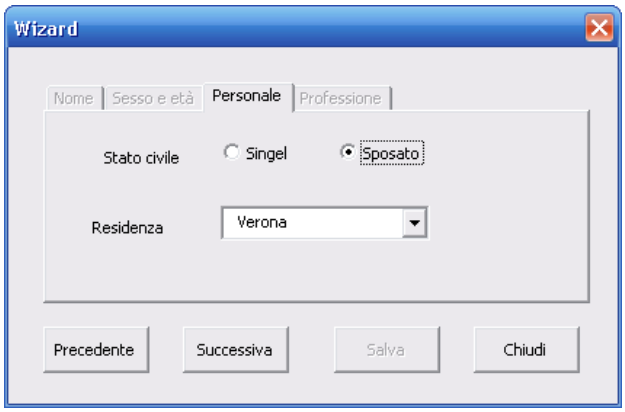


Fig. 6

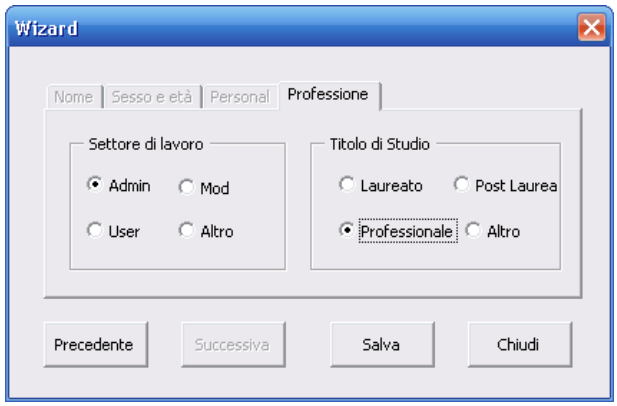


Fig. 7

Esempio: Procedura guidata multipage

```
Private Sub UserForm_Initialize()
    Dim i As Integer
'impostare la Caption per tutte le pagine
MultiPage1.Pages(0).Caption = "Nome"
MultiPage1.Pages(1).Caption = "Sesso e età"
MultiPage1.Pages(2).Caption = "Personale"
MultiPage1.Pages(3).Caption = "Professione"
'impostare le Caption dei Frame in pagina 4
MultiPage1.Pages(3).Frame1.Caption = "Settore di lavoro"
MultiPage1.Pages(3).Frame2.Caption = "Titolo di Studio"

'riempire il ListBox dell'età in pagina 2
With MultiPage1.Pages(1).ListBox1
For i = 1 To 100
.AddItem i & " anni"
Next i
End With

'riempire il ComboBox residenza di pagina 3
myArray = Array("Milano", "Verona", "Roma", "Bari", "Firenze", "Torino", "Palermo",
"Genova", "Bologna", "Napoli", "Salerno", "Venezia", "Udine", "Trento", "Trieste", "Ancona",
"Messina")
MultiPage1.Pages(2).ComboBox1.List = myArray

MultiPage1.Pages(0).Enabled = True
MultiPage1.Pages(1).Enabled = False
MultiPage1.Pages(2).Enabled = False
MultiPage1.Pages(3).Enabled = False
'seleziona la prima pagina
MultiPage1.Value = 0
End Sub

Private Sub MultiPage1_Change()
    Select Case MultiPage1.Value
'prima pagina
Case 0
CommandButton1.Enabled = False
CommandButton2.Enabled = True
CommandButton3.Enabled = False
'ultima Pagina
Case MultiPage1.Pages.Count - 1
CommandButton1.Enabled = True
CommandButton2.Enabled = False
CommandButton3.Enabled = True
'altre Pagine
Case Else
CommandButton1.Enabled = True
CommandButton2.Enabled = True
CommandButton3.Enabled = False
End Select
End Sub
```

```
Private Sub commandbutton3_Click()
With MultiPage1.Pages(MultiPage1.Pages.Count - 1)
If OptionButton5.Value = False And OptionButton6.Value = False And OptionButton7.Value
= False And OptionButton8.Value = False Then
MsgBox "Inserisci il settore di lavoro"
Exit Sub
ElseIf OptionButton9.Value = False And OptionButton10.Value = False And
OptionButton11.Value = False And OptionButton12.Value = False Then
MsgBox "Inserisci il titolo di studio"
Exit Sub
End If
End With
End Sub

Private Sub commandbutton1_Click()
Select Case MultiPage1.Value
Case 1
MultiPage1.Pages(1).Enabled = False
MultiPage1.Pages(0).Enabled = True
MultiPage1.Value = 0
Case 2
MultiPage1.Pages(2).Enabled = False
MultiPage1.Pages(1).Enabled = True
MultiPage1.Value = 1
Case 3
MultiPage1.Pages(3).Enabled = False
MultiPage1.Pages(2).Enabled = True
MultiPage1.Value = 2
End Select
End Sub
```

```
Private Sub commandbutton2_Click()
   Select Case MultiPage1.Value
Case 0
If TextBox1.Value = "" Then
MsgBox "Inserisci il nome"
Exit Sub
Else
MultiPage1.Pages(0).Enabled = False
MultiPage1.Pages(1).Enabled = True
MultiPage1.Value = 1
End If

Case 1
If OptionButton1.Value = False And OptionButton2.Value = False Then
MsgBox "Seleziona il sesso"
Exit Sub
ElseIf ListBox1.ListIndex = -1 Then
MsgBox "Seleziona l'età"
Else
MultiPage1.Pages(1).Enabled = False
MultiPage1.Pages(2).Enabled = True
MultiPage1.Value = 2
End If

Case 2
If OptionButton3.Value = False And OptionButton4.Value = False Then
MsgBox "Seleziona lo stato coniugale"
Exit Sub
ElseIf ComboBox1.Value = "" Then
MsgBox "Seleziona la città"
Else
MultiPage1.Pages(2).Enabled = False
MultiPage1.Pages(3).Enabled = True
MultiPage1.Value = 3
End If
End Select
End Sub

Private Sub CommandButton4_Click()
Unload Me
End Sub
```

Il Controllo TabStrip

Un controllo TabStrip viene utilizzato per visualizzare contenuti diversi in ogni scheda per la stessa serie di controlli. Un TabStrip è una raccolta di schede in cui ogni scheda contiene una serie di controlli. Per impostazione predefinita, un controllo TabStrip ha 2 Tabs, per aggiungere delle schede, si deve fare clic col destro del mouse sulla scheda e selezionare Nuova Pagina (selezionare Elimina pagina per eliminare un Tab).

Aggiungi o rimuovi una scheda

Per aggiungere una scheda si utilizzoa il *Metodo Add* che ha questa sintassi:

Set t = TabStrip1.Tabs.Add (tabname, tabCaption, tabIndex) dove

tabIndex (opzionale) è un numero intero che specifica la posizione della linguetta da inserire, si inizia da 0 per la prima posizione, mentre
tabname imposta il nome per la scheda e
tabCaption imposta la Caption, entrambi sono opzionali

Set t = TabStrip1.Tabs.Add ("TAB4", "newtab", 1) Questo codice aggiunge una nuova scheda con il nome *TAB4* e Caption *newtab*, come seconda scheda (cioè seconda posizione nell'ordine di tabulazione).

TabStrip1.Tabs.Add "Tab3"
TabStrip1.Tabs (2) Caption = "Prova": Questi 2 codici aggiungono una nuova (terza) scheda denominata *Tab3* e impostano la Caption a *Prova*

Per rimuovere una scheda si usa la sintassi: *TabStrip1.Tabs.Remove (tabCaption)* .
Esempio: *TabStrip1.Tabs.Remove ("Prova")*, questo codice rimuove la scheda con Caption *Prova*

Differenza tra un controllo Multipage e TabStrip

Un controllo multipage è un contenitore per i controlli, simile a un frame e ogni pagina ha un insieme separato di controlli e selezionando una pagina si nascondono le altre pagine del controllo, mentre un controllo TabStrip contiene un insieme coerente di controlli in tutte le schede e il contenuto dei controlli cambia quando è selezionata una scheda diversa ma la visibilità o disposizione dei comandi rimangono stesso.

Selezione di una scheda

Per modificare o impostare le proprietà di una scheda a run-time, si deve identificare il *Tab* nel controllo *TabStrip*, che può essere fatto in diversi modi, si può usare la proprietà *SelectedItem* del controllo TabStrip che indica che è selezionata la scheda *Tab*. Si ricorda che per selezionare una scheda si deve impostare la proprietà *Value* del controllo TabStrip. I valori partono da 0, e la prima scheda in un controllo TabStrip avrà un valore 0, il valore della seconda sarà 1, e così via. Per accedere a una singola scheda in un controllo TabStrip, si possono usare i seguenti metodi

- *Numeric Index* (Indice numerico): Con indice 0 si riferisce alla prima pagina, con indice 1 alla seconda e così via. Il codice per impostare la Caption è : *TabStrip1.Tabs (Index).Caption*
- *Metodo Item* (utilizzando l'insieme Tabs): Con Indice 0 si riferisce alla prima pagina, con indice 1 alla seconda pagina e così via.Il codice per impostare la Caption è : *TabStrip1.Tabs.Item (ItemIndex).Caption*
- Tab Name : Il codice per impostarela Caption è: *TabStrip1.Tabs ("tabname").Caption* o ppure *TabStrip1.Tabs.Item ("TabName"). Caption*
- *Tab Object* : Il codice per impostare la Caption è: *TabStrip1.TabName.Caption*
- *Proprietà SelectedItem* : Il codice per impostare la Caption è : *TabStrip1.SelectedItem.Caption*

Esempio: Cambiare impostare e proprietà di ogni scheda di un controllo TabStrip, utilizzando diversi metodi di selezione delle tabulazioni.

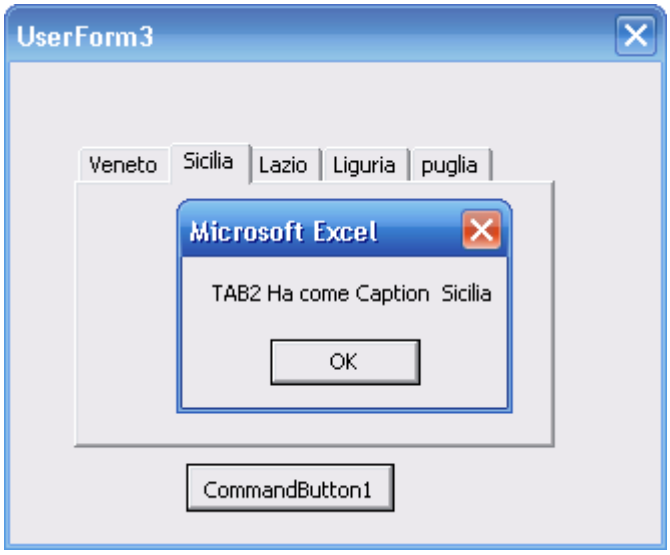


Fig. 8

```
Private Sub UserForm_Initialize()
TabStrip1.Tabs(0).Caption = "Veneto"
TabStrip1.Tabs.Item(1).Caption = "Sicilia"
TabStrip1.Tabs("Tab3").Caption = "Lazio"
TabStrip1.Tab4.Caption = "Liguria"
TabStrip1.Value = 4
TabStrip1.SelectedItem.Caption = "puglia"
End Sub

Private Sub CommandButton1_Click()
    Dim i As Integer
For i = 0 To TabStrip1.Tabs.Count - 1
MsgBox TabStrip1.Tabs(i).Name & " Ha come Caption  " & TabStrip1.Tabs(i).Caption
Next i
End Sub
```

Esempio: Come lavorare con un controllo TabStrip e Tabs - utilizzando un TabStrip e le sue schede per caricare dati da un foglio di lavoro e aggiornare l'intervallo dai dati della scheda.

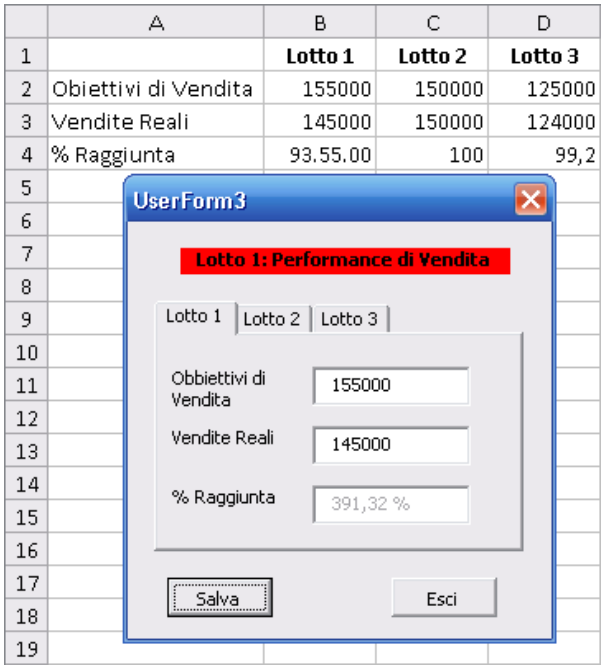


Fig. 9

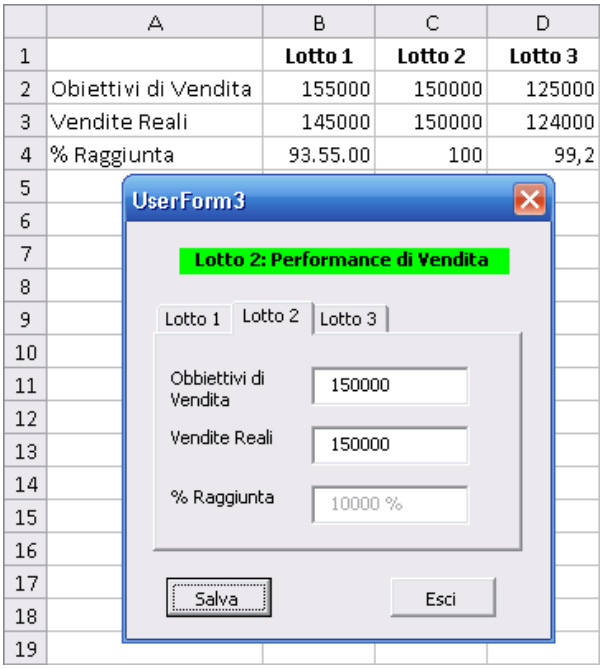


Fig. 10

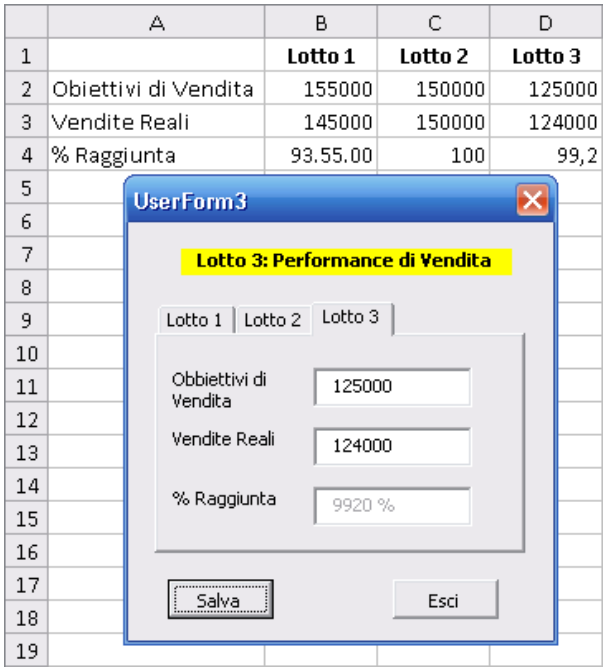


Fig. 11

```
Private Sub UserForm_Initialize()
With TabStrip1
.Tabs(0).Caption = "Lotto 1"
.Tabs(1).Caption = "Lotto 2"
.Tabs(2).Caption = "Lotto 3"
End With

TextBox1.Value = Foglio1.Range("B2").Value
TextBox2.Value = Foglio1.Range("B3").Value
TextBox3.Value = Round(Foglio1.Range("B4").Value * 100, 2) & " %"
TabStrip1.Value = 0
Label4.Caption = "Lotto 1: Performance di Vendita"
Me.Label4.BackColor = RGB(255, 0, 0)
Me.Label4.Font.Bold = True
Me.Label4.TextAlign = fmTextAlignCenter
TextBox3.Enabled = False
End Sub

Private Sub TabStrip1_Change()
    Dim n As Integer
n = TabStrip1.SelectedItem.Index

Select Case n
Case 0
Label4.Caption = "Lotto 1: Performance di Vendita"
Me.Label4.BackColor = RGB(255, 0, 0)
TextBox1 = Foglio1.Range("B2").Value
TextBox2 = Foglio1.Range("B3").Value
TextBox3 = Round(Foglio1.Range("B4").Value * 100, 2) & " %"

Case 1
Label4.Caption = "Lotto 2: Performance di Vendita"
Me.Label4.BackColor = RGB(0, 255, 0)
TextBox1 = Foglio1.Range("C2").Value
TextBox2 = Foglio1.Range("C3").Value
TextBox3 = Round(Foglio1.Range("C4").Value * 100, 2) & " %"

Case 2
Label4.Caption = "Lotto 3: Performance di Vendita"
Me.Label4.BackColor = RGB(255, 255, 0)
TextBox1 = Foglio1.Range("D2").Value
TextBox2 = Foglio1.Range("D3").Value
TextBox3 = Round(Foglio1.Range("D4").Value * 100, 2) & " %"
End Select
End Sub
```

```
Private Sub CommandButton1_Click()
    Dim n As Integer
n = TabStrip1.SelectedItem.Index

Select Case n
Case 0
If IsNumeric(TextBox1.Value) And TextBox1.Value > 0 And IsNumeric(TextBox2.Value) Then
Foglio1.Range("B2").Value = TextBox1.Value
Foglio1.Range("B3").Value = TextBox2.Value
TextBox3.Value = Round((TextBox2.Value / TextBox1.Value) * 100, 2) & " %"
Else
TextBox3.Value = ""
End If

Case 1
If IsNumeric(TextBox1.Value) And TextBox1.Value > 0 And IsNumeric(TextBox2.Value) Then
Foglio1.Range("C2").Value = TextBox1.Value
Foglio1.Range("C3").Value = TextBox2.Value
TextBox3.Value = Round((TextBox2.Value / TextBox1.Value) * 100, 2) & " %"
Else
TextBox3.Value = ""
End If

Case 2
If IsNumeric(TextBox1.Value) And TextBox1.Value > 0 And IsNumeric(TextBox2.Value) Then
Foglio1.Range("D2").Value = TextBox1.Value
Foglio1.Range("D3").Value = TextBox2.Value
TextBox3.Value = Round((TextBox2.Value / TextBox1.Value) * 100, 2) & " %"
Else
TextBox3.Value = ""
End If
 End Select
End Sub
```

I Controlli in una Userform

Un oggetto Userform può contenere controlli come possiamo vedere nelle finestre di dialogo normalmente mostrate da Excel o altri programmi. I controlli sono gli elementi di una finestra che consentono all'utente di interagire con il programma. Nella lezione precedente abbiamo fatto una panoramica su questo argomento, adesso vediamo di approfondire elencando i vari controlli disponibili in maniera più dettagliata.

I controlli disponibili in VBA sono:

Label : Corrisponde a Etichetta, inserisce un oggetto di tipo label1 e svolge la funzione di etichetta descrittiva per quei controlli che non ne hanno una propria. E' inoltre possibile utilizzare le etichette per mostrare il valore di una variabile. Si può modificare il testo mostrato di una etichetta (label1 nel nostro caso) modificando la proprietà Captino, questo è possibile via codice oppure agendo nella finestra delle proprietà, generalmente questo oggetto viene usato nei casi in cui non si debba modificarne il testo, ma semplicemente come elemento descrittivo di un altro controllo

Casella di testo : Inserisce un oggetto di tipo Textbox, questo oggetto permette l'inserimento di testo da parte dell'utente, viene utilizzato per ogni tipo di campo che può essere rappresentato come testo, nomi, date, numeri valute e inoltre può accettare e mostrare più righe di testo, infatti quando necessario compare una barra di scorrimento verticale che permette di scorrere il testo contenuto

Cornice : Inserisce un oggetto di tipo Frame, questo oggetto non ha nessuna funzione specifica, se non quella di raggruppare logicamente un insieme di controlli. Si utilizza solitamente per raggruppare pulsanti di opzione, caselle di controllo e pulsanti interruttore. Usate questo controllo quando volete far risaltare all'utente che i controlli contenuti nella cornice sono in qualche modo collegati tra loro

Pulsante di comando : Inserisce un oggetto di tipo CommandButton, il classico pulsante di standard di Windows che si attiva quando l'utente fa clic su di esso. L'azione svolta al clic sul controllo dipende dal codice che viene messo nell'evento, può compiere operazioni di chiusura *(Close)* di nascondimento*(Hide)* , di aggiornamento dei dati, di verifica e così via

Casella di Controllo : Inserisce un oggetto di tipo CheckBox, è composto da un quadrato che contiene un segno di spunta nel caso in cui sia selezionato e da un'etichetta di descrizione. Utilizzate questo controllo per le opzioni che possono avere un valore di tipo Vero o Falso. Può restituire valori True (Vero), quando la casella è selezionata o False (Falso), se la casella non è selezionata, inoltre la presenza di più caselle di controllo non si escludono a vicenda

Pulsante di opzione : Inserisce un oggetto di tipo OptionButton, è composto da un pulsante tondo che contiene una pallina nera nel caso sia selezionato e da un'etichetta di descrizione. Viene usato con una serie di pulsanti di opzione per consentire all'utente una selezione di elementi che si escludono a vicenda

Pulsante Interruttore : Inserisce un oggetto di tipo ToggleButton, mostra lo stato del pulsante Vero o Falso, Acceso o Spento, allo stesso modo di una casella di controllo solo che ha l'aspetto di un pulsante in posizione "Su" o "Giù"

Casella di riepilogo : Inserisce un oggetto di tipo ListBox e mostra una serie di dati presenti nel foglio in cui l'utente può fare una scelta, inoltre la proprietà *MultiSelect* controlla la possibilità di effettuare la scelta di un solo valore o di più elementi della lista

Casella combinata : Inserisce un oggetto di tipo ComboBox, questo controllo è la combinazione grafica di una casella di testo (TextBox) e di una casella di riepilogo (ListBox) la casella di testo permette la digitazione di dati con la possibilità di suggerire una serie di valori tra cui scegliere attraverso la lista a discesa. Tramite questo controllo è possibile consentire all'utente di inserire un valore non presente nella lista, oppure costringerlo a scegliere un valore tra quelli presenti nella lista. Per poter effettuare questa impostazione bisogna modificare la proprietà *Style* nella finestra delle proprietà

Schede : Inserisce un oggetto di tipo TabStrip, questo controllo consiste in una singola area in cui è possibile inserire altri controlli, è costituito da una serie (modificabile a piacere) di pulsanti di tabulazione. Questo controllo è simile all'oggetto cornice, in quanto l'area non cambia, ma tramite i pulsanti di tabulazione è possibile mostrare dati di diverse categorie. Per esempio in un'anagrafica di un cliente è possibile avere un tabulatore in cui compaiono i dati generali (indirizzo, telefono etc.) un altro tabulatore mostrerà i dati bancari e così via

Pagine : Inserisce un oggetto di tipo MultiPage, questo controllo è simile al controllo Schede (TabStrip) inoltre ogni pagina di questo controllo può contenere diversi controlli distinti. E' da utilizzare quando ogni pagina deve avere un contenuto differente

Barra di scorrimento : Inserisce un oggetto di tipo ScrollBar e permette di scorrere i valori contenuti in una finestra

Casella di selezione : Inserisce un oggetto di tipo SpinButton, è una particolare categoria di casella di testo che permette l'inserimento facilitato e la modifica di dati compresi in un certo intervallo. Viene utilizzato per l'inserimento di valori numerici, date o valori in sequenza e si utilizza insieme ad un controllo etichetta(Label) o casella di testo (TextBox). In pratica facendo clic sulla freccia "su" o "giù" il valore viene incrementato o diminuito

Immagine : Inserisce un oggetto di tipo Image, e permette di inserire un'immagine all'interno della finestra in uno dei seguenti formati *.bmp, *.cur, *.jpeg, *.gif, *.ico

RefEdit : Inserisce un oggetto di tipo RefEdit, questa è una speciale casella di testo che consente di inserire e selezionare intervalli sui fogli di lavoro di Excel

Ogni controllo è un oggetto con proprietà, metodi ed eventi specifici esattamente come le finestre (Userform) che li contiene, è possibile impostare le proprietà dei controlli via codice oppure utilizzando la finestra Proprietà dell'editor di Visual Basic. Una volta aggiunto un controllo alla finestra, potete effettuare una serie di operazioni come: copiare l'oggetto, ridimensionarlo, spostarlo, cancellarlo, modificarne la formattazione, il tipo di carattere, il colore e modificare tutte le sue proprietà. Avrete sicuramente la necessità di fare queste operazioni nel momento in cui perfezionerete la vostra tecnica di programmazione delle finestre di dialogo. Per esempio potete decidere che una casella di testo è troppo ampia o troppo stretta per il testo a cui è destinata, così ne modificherete le dimensioni. Oppure un pulsante di comando con la scritta CommandButton7 avrebbe un significato a dir poco "oscuro" per l'utente che userà la vostra finestra di dialogo.

Per poter inserire un controllo nella nostra Userform dovete fare clic sul pulsante *della Casella degli strumenti* corrispondente al controllo che volete aggiungere nella finestra, in questo modo il puntatore del mouse si trasforma in una croce sottile quando viene posizionato sulla Userform. Posizionate il puntatore a croce sulla finestra nel punto in cui volete inserire il controllo, tenete presente che il punto che scegliete corrisponderà all'angolo superiore sinistro. Fate clic e tenete premuto il pulsante sinistro del mouse, Trascinate ora a destra e in basso fino a che il controllo non raggiunge le dimensioni desiderate, quindi rilasciate il pulsante, a questo punto l'editor di Visual basic inserisce il controllo e il puntatore del mouse ritorna ad essere la freccia standard.

Se proviamo ad inserire una casella di testo (TextBox), con le istruzioni sopra riportate ci comparirà una finestra come quella sotto riportata

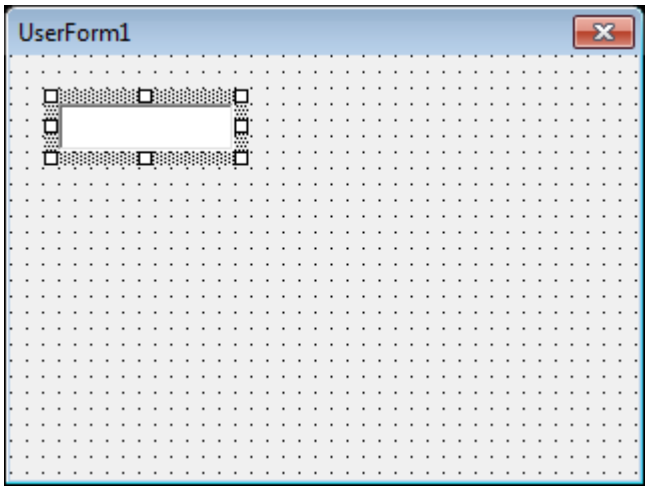


Fig. 1

Avrete notato che il controllo è circoscritto da dei piccoli quadratini bianchi che ci permettono di ridimensionare il controllo stesso, posizionandoci col cursore del mouse su uno di essi il puntatore si trasforma in una doppia freccia, che indica la direzione nella quale è possibili ridimensionare l'oggetto. Inoltre posizionandoci sopra al controllo il cursore cambia aspetto e diventa a 4 frecce, in questa condizione è possibile spostare il controllo nella posizione voluta. Ricordate anche che ogni controllo aggiunto alla finestra di dialogo deve avere un nome univoco, a questo pensa già l'editor di VBA, se inseriamo un controllo CommandButtons gli viene assegnato automaticamente il nome del controllo seguito da un numero [es. CommandButtons1] questo numero progressivo viene incrementato automaticamente dall'editor ogni volta che si inserisce un controllo dello stesso tipo.

Modificare i controlli

Una volta che avete aggiunto un controllo alla finestra potete effettuare una serie di operazioni come: copiare l'oggetto, ridimensionarlo, spostarlo, cancellarlo, modificarne la formattazione etc. Avrete sicuramente la necessità di compiere queste operazioni nel momento in cui perfezionerete la vostra tecnica di progettazione delle finestre di dialogo. Per esempio potete decidere che una casella di testo è troppo grande o troppo stretta per il testo che è destinata a ricevere, così necessita di una modifica delle dimensioni, oppure inserire un controllo *CommandButon5* avrebbe un significato oscuro per l'utente che userà il programma.

Per modificare un controllo occorre prima selezionarlo facendo clic sopra al controllo stesso e per modificarlo dobbiamo ricorrere alla finestra delle proprietà. Inseriamo ora per esempio 3 controlli, un CommandButton, una TextBox e una Label nella nostra form che adesso si presenta così

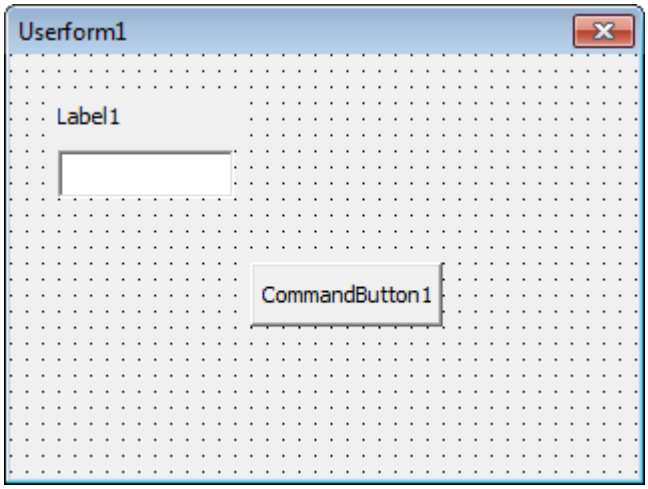


Fig. 2

Presentare una Userform in questo modo non è certamente accattivante, dobbiamo cercare di dare una indicazione visiva all'utente sullo scopo della Userform. Iniziamo modificando i 3 controlli inseriti, trascinandoli nella posizione giusta e utilizzando la finestra delle proprietà per modificare il valore *caption*, modifichiamo anche il titolo della form inserendo la dicitura "Esempio di inserimento", come etichetta della Label usiamo "Inserisci un valore numerico" e il pulsante di comando diventa "Verifica". Come già detto tutte queste modifiche le effettuiamo selezionando il controllo e andando a modificare il valore del campo Caption, come evidenziato dalla *freccia rossa in figura 3*

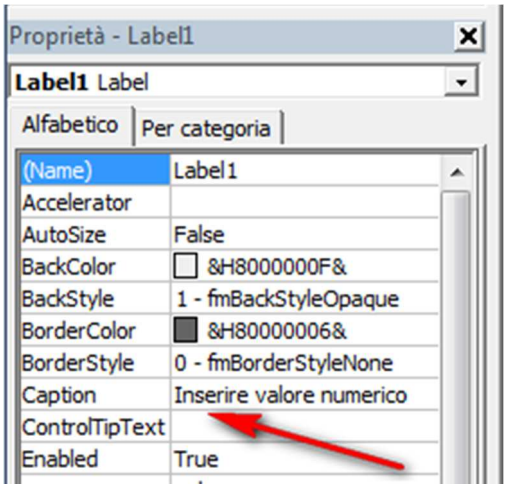


Fig. 3

Una volta terminata la modifica dei vari controlli e ridimensionata la finestra, otteniamo una form come la seguente

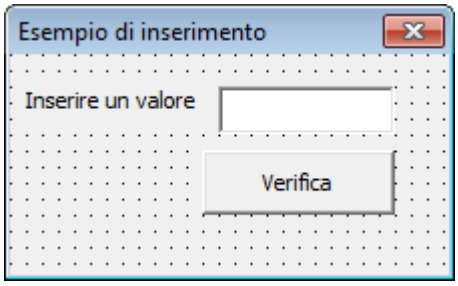


Fig. 4

Possiamo ipotizzare che questa finestra attenda l'inserimento di un valore nella TextBox e al tempo stesso verifichi che sia stato inserito un valore numerico prima di salvarlo in una cella del foglio di lavoro. In questi casi è sempre opportuno fare la verifica del testo inserito per evitare ulteriori errori di runtime nel proseguo del programma. E' possibile fare verifiche del genere in due modi, uno sfruttando l'evento **Change** della TextBox e l'altro con opportuno codice quando clicchiamo sul pulsante di verifica prima di eseguire il salvataggio. Vediamo entrambi i metodi, prima però aggiungiamo un'altra Label alla nostra form che ci torna utile per rimandare un avviso in caso di errore. La possiamo inserire in questo modo

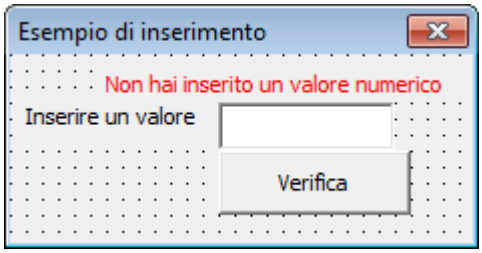


Fig. 5

Come potete vedere è stato cambiato anche il colore del testo agendo sul campo **ForeColor** della finestra delle proprietà come si vede in figura

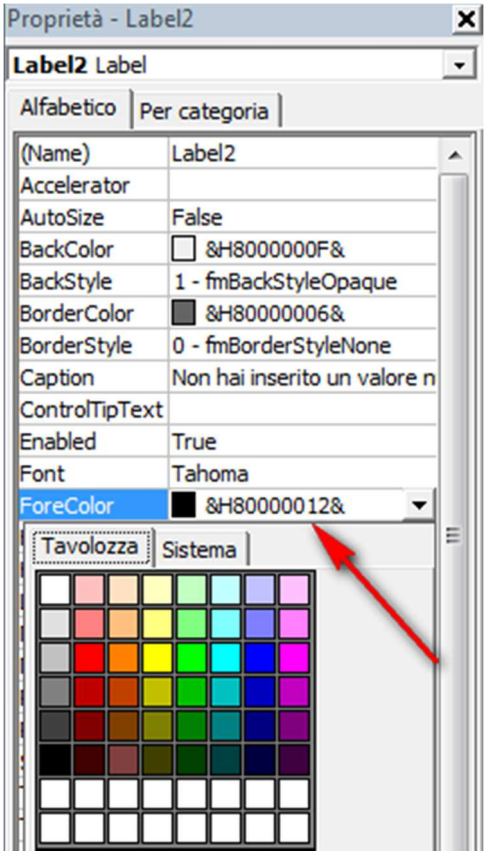


Fig. 6

A questo punto dobbiamo inserire il codice adatto nell'evento Change e per raggiungere questo evento facciamo doppio clic sulla TextBox e nella finestra del codice ci compare l'evento TextBox1_Change(). Abbiamo già detto che questo evento si scatena quando sente un cambiamento all'interno del campo stesso, per cui se inseriamo un codice come il seguente

```
Private Sub Textbox1_Change()
    If IsNumeric(Textbox1.Value) Then
     Label2.Visible = False
    Else
        Label2.Visible = True
TextBox1.Value=""
    End If
End Sub
```

Se inseriamo un valore numerico la Label2 **NON** diventa visibile, mentre in caso di valore **NON** numerico la Label2 diventa visibile e la routine termina cancellando il valore nella TextBox. Se eseguiamo questa routine ci viene riportata la form in questo modo

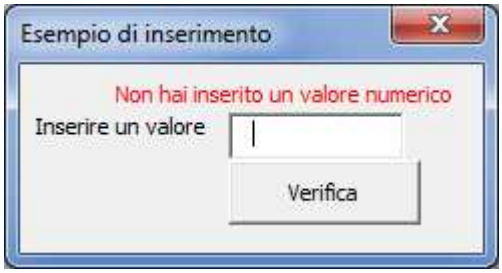


Fig. 7

Come potete vedere la Label2 è visibile, mentre noi vogliamo che diventi visibile in caso di errore nell'inserimento del tipo di dato richiesto. La Label2 è visibile in quanto non abbiamo inizializzato la form, nella lezione precedente abbiamo visto gli eventi **Initialize** e **Activate** delle Userform, nel nostro caso utilizziamo l'evento Initialize per nascondere la Label2 e farla poi apparire in caso di errore tramite il ciclo If presente nell'evento **Change** della TextBox, pertanto andremo ad aggiungere questo codice

```
Private Sub UserForm_Initialize()
Label2.Visible = False
End Sub
```

In questo modo la finestra di dialogo ci compare in questo modo

Fig. 8

A questo punto se inseriamo un valore di tipo stringa nel campo della TextBox ci viene mostrata la Label di avviso errore di inserimento

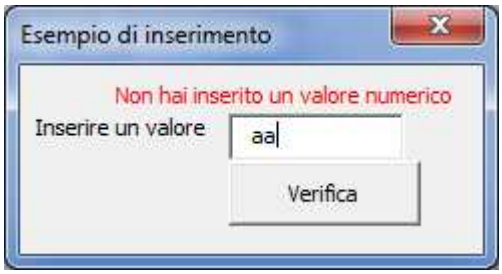


Fig. 9

Per mostrare la Label2 e il testo inserito nel campo della TextBox è stata remmata (è stata resa come un commento e il VBA salta i commenti) un'istruzione nella routine

```
Private Sub Textbox1_Change()
    If IsNumeric(TextBox1.Value) Then
     Label2.Visible = False
    Else
        Label2.Visible = True
' TextBox1.Value = ""
    End If
End Sub
```

Come vedete è stato posto un apostrofo prima dell'istruzione TextBox1.Value = "" che è quella che si occupa di svuotare il campo. A questo punto abbiamo visto il primo metodo di verifica dei dati, l'altro, come abbiamo precedentemente accennato, viene eseguito premendo il pulsante di "Verifica" prima di eseguire la scrittura del valore della TextBox nel foglio di lavoro. Lo possiamo fare in questo modo

```
Private Sub CommandButton1_Click()
    If IsNumeric(Textbox1.Value) Then
        Range("A1") = Textbox1.Value
        Unload Me
    Else
        MsgBox "Il Valore inserito è errato"
    End If
End Sub
```

Se eseguiamo questo codice (ovviamente dobbiamo togliere il codice nell'evento TextBox1_Change altrimenti scatta anche quello) in caso di inserimento di un valore stringa ci rimanda questo avviso

Fig. 10

I controlli Label, TextBox e CommandButton

Il controllo Label

Il controllo Label serve per visualizzare un testo non modificabile dall'utente e viene utilizzato per descrivere altri controlli, ed è spesso usato per descrivere un TextBox o per visualizzare delle informazioni. Questo tipo di controllo deve essere inserito in un controllo UserForm e dato che è molto utilizzato, è inutile assegnare un nome significativo a ogni sua istanza, a meno che non la si debba modificare durante l'esecuzione del programma. Si utilizzano principalmente le seguenti proprietà: *Caption* per visualizzare il testo, *Left* e *Top* per posizionarlo, *TextAlign* per allineare il testo all'interno dell'etichetta, *Font* per lo stile e il formato del testo, *BackColor* e *ForeColor* per i colori di sfondo e del testo. È possibile formattare una etichetta sia nella finestra Proprietà o mediante un'istruzione VBA come: *Label1.Caption = "Inserire la quantità richiesta"*, oppure utilizzando una dichiarazione *With* come illustrato di seguito.

Esempio: Cliccando sul pulsante di una Form verrà formattato il testo della Label

```
Private Sub CommandButton1_Click()
With Label1
.Caption = "Inserire il testo"
'Allineamento del testo impostato al centro
.TextAlign = fmTextAlignCenter
.WordWrap = True
'Impostare il tipo di carattere
.Font.Name = "Arial"
.Font.Size = 12
.Font.Italic = True
'impostare a giallo e il colore del testo e a rosso il colore di sfondo
.ForeColor = RGB(255, 255, 0)
.BackColor = RGB(255, 0, 0)
End With
End Sub
```

Il Controllo CommandButton

Un *CommandButton* viene in genere utilizzato per eseguire una macro tramite l'evento Click. Se entrate in VBE è possibile accedere all'evento Click facendo doppio clic sul CommandButton, oppure è possibile selezionare il nome del CommandButton (nel modulo di codice per il Form utente) nella casella a discesa nella finestra del codice a sinistra e poi selezionare e poi cliccare nella tendina in alto a destra. Cliccando sul CommandButton verrà eseguito il codice che viene inserito nell'evento *Click* oppure è possibile inserire il nome di una macro da eseguire

Esempio: Usando l'evento Click si può chiudere la Form

```
Private Sub CommandButton2_Click ()
MsgBox "Chiusura UserForm !"
Unload Me
End Sub
```

Esempio: Cliccando sul pulsante di comando si esegue un'altra macro per svuotare tutti i controlli della Form

```
Private Sub CommandButton1_Click ()
ClearForm
End Sub

Private Sub ClearForm ()
Textbox1.value = ""
ComboBox1.Value = ""
CheckBox1.Value = False
OptionButton1.Value = False
End Sub
```

Il Controllo TextBox

Un controllo TextBox accetta dati da parte dell'utente. Oltre alle proprietà comuni menzionate in precedenza, le proprietà chiave includono:

Proprietà AutoTab: Restituisce un valore booleano (True o False) che specifica se la scheda passa automaticamente al controllo successivo nell'ordine di tabulazione dopo il numero massimo di caratteri determinato dalla struttura *MaxLength* all'entrata nel TextBox da parte dell'utente, mentre invece il valore False (che è di default) indica che lo spostamento al controllo successivo avviene manualmente quando l'utente preme il tasto Tab.

Proprietà EnterKeyBehavior: Restituisce un valore booleano (True o False) che determina l'effetto quando un utente preme il tasto Invio in un TextBox. Se la proprietà *MultiLine* è impostata su True, il valore indica la creazione di una nuova linea premendo Invio mentre il valore False (Default) si passa al controllo successivo nell'ordine di tabulazione. Se MultiLine proprietà è impostata su False, il Focus è sempre spostato al controllo successivo nell'ordine di tabulazione ignorando la proprietà EnterKeyBehavior

Proprietà MaxLength: Specifica il numero massimo di caratteri che possono essere inseriti in una TextBox. Specificando un valore di 0 indica che non vi è alcun limite massimo.

Proprietà MultiLine: Restituisce un valore booleano (True o False) che determina se il testo viene visualizzato in più righe o meno, nella TextBox. True indica che il testo viene visualizzato in più righe, e questo è anche il valore di default.

Proprietà PasswordChar: Specifica quali caratteri si devono visualizzare nella TextBox al posto dei caratteri effettivamente inseriti o digitati dall'utente. Questa proprietà è utile per proteggere i codici di sicurezza sensibili, o per convalidare un utente prima di consentire di procedere ulteriormente.

Proprietà ScrollBars: Specifica se un TextBox ha barre di scorrimento verticali e/o orizzontali, o nessuna. Ci sono 4 impostazioni "auto-esplicativo":

- *fmScrollBarsNone* (valore 0) - questa è l'impostazione di default
- *fmScrollBarsHorizontal* (valore 1)
- *fmScrollBarsVertical* (valore 2)
- *fmScrollBarsBoth* (valore 3).

L'impostazione *fmScrollBarsNone* non visualizza nessuna barra di scorrimento e se la proprietà *AutoSize* è impostata su True, non comparirà nessuna barra di scorrimento perché il TextBox si allarga per accogliere il testo o di dati aggiuntivi.

Se *WordWrap* è impostata su True, non comparirà nessuna barra di scorrimento orrizzontale nel TextBox. Le impostazioni della barra di scorrimento *fmScrollBarsHorizontal* o *fmScrollBarsBoth*, visualizzano una barra di scorrimento orizzontale in una TextBox a linea singola se il testo è più lungo rispetto alla capienza della casella. Le impostazioni della barra di scorrimento*fmScrollBarsVertical* o *fmScrollBarsBoth*, visualizzano una barra di scorrimento verticale in una TextBox a più righe se il testo è più lungo rispetto alla capienza della casella e WordWrap è impostata su True.

Per visualizzare una barra di scorrimento orizzontale in un TexBox multilinea, l'impostazione della barra di scorrimento dovrebbe essere*fmScrollBarsHorizontal*, e WordWrap dovrebbe essere impostata su False e il testo dovrebbe essere più lungo rispetto alla capienza della casella.

NB *Una barra di scorrimento orizzontale (o verticale) è visibile solo se il controllo ha spazio sufficiente per includere la barra di scorrimento sotto o al bordo destro della sua casella*.

Proprietà Text: Il testo in un TextBox viene restituito o impostato da questa proprietà. Un valore assegnato alla proprietà Text viene assegnato automaticamente al valore della proprietà, e viceversa.

Esempio: Aggiungere un TextBox in un Form e formattarlo utilizzando il codice VBA

```
Private Sub CommandButton1_Click()
Dim box_1 As MSForms.TextBox
Set box_1 = Controls.Add("Forms.TextBox.1", "Esempio 1")
With box_1
.Font.Name = "Times New Roman"
.Font.Size = 10
.TextAlign = fmTextAlignLeft
.Width = 100
.Height = 50
.Left = 50
.Top = 75
.MultiLine = True
.WordWrap = True
.AutoSize = False
.ScrollBars = 2
.SetFocus
End With
End Sub
```

Esempio: Impostare la proprietà Enabled di una TextBox per impedire all'utente di digitare direttamente nella casella di testo che deve essere riempito solo per l'opzione selezionata dall'utente nel ListBox

```
Private Sub UserForm_Initialize()
With ListBox1
For i = 1 To 10
.AddItem i
Next i
.ControlTipText = "Selezionare un numero dal Listbox per inserirlo nel TextBox."
End With
Me.TextBox1.Enabled = False
End Sub
Private Sub ListBox1_Click()
TextBox1.Text = ListBox1.Value
End Sub
```

Esempio: Utilizzare una TextBox per impostare una password, in questo esempio usiamo la proprietà *PasswordChar* per controllare se il nome utente e la password, sono autorizzati a procedere e caricare UserForm1 viene caricato.

```
Private Sub UserForm_Initialize()
TextBox2.MaxLength = 5
TextBox2.PasswordChar = "*"
TextBox2.BackColor = RGB(255, 255, 0)
End Sub

Private Sub CommandButton1_Click()
Dim password As String
If TextBox1.Text = "Pippo" And TextBox2.Text = "123456" Then
password = "True"
ElseIf TextBox1.Text = "Topolino" And TextBox2.Text = "987654" Then
password = "True"
ElseIf TextBox1.Text = "Paperino" And TextBox2.Text = "369852" Then
password = "True"
End If
If password = "True" Then
MsgBox "Password Esatta Puoi Continuare"
Unload Me
UserForm1.Show
Else
MsgBox "Paasword o Username errato. Riprova"
TextBox1.Text = vbNullString
TextBox2.Text = vbNullString
TextBox1.SetFocus
End If
End Sub
```

I controlli ListBox e ComboBox

I controlli ListBox e ComboBox permettono di visualizzare una lista di opzioni selezionabili dall'utente. La ragione principale per usare una casella combinata (**ComboBox**) o una casella di riepilogo (**ListBox**) è quella di poter visualizzare un elenco di elementi, dove nella casella di riepilogo (ListBox) è un elenco a discesa e gli elementi della lista sono sempre visibili mentre nel ComboBox la lista è "a scomparsa", ovvero è visibile soltanto se l'utente fa clic sulla freccia verso il basso a destra del controllo. Le caselle combinate sono così chiamate perché sono composte da due parti, una porzione di testo che ricorda il controllo TextBox e un elenco che riporta ad una Listbox e "combinano" le caratteristiche che si trovano in entrambe le caselle di testo (TextBox) e le caselle di riepilogo (ListBox) e sono anche comunemente chiamate "elenchi a discesa".

In sostanza la ComboBox è un elenco a discesa in cui la voce selezionata dall'elenco è visibile nell'area di testo, mentre i valori della lista sono visibili solo cliccando sul menu a tendina, mentre un ListBox mostra un certo numero di valori con o senza una barra di scorrimento, inoltre in una ComboBox, è visibile una sola voce senza utilizzare la tendina a discesa, mentre in un ListBox sono visibili molti più elementi. In una ComboBox è possibile selezionare solo una voce dall'elenco, mentre invece in un ListBox è possibile selezionare più opzioni dall'elenco. La rappresentazione grafica delle due caselle è la seguente:

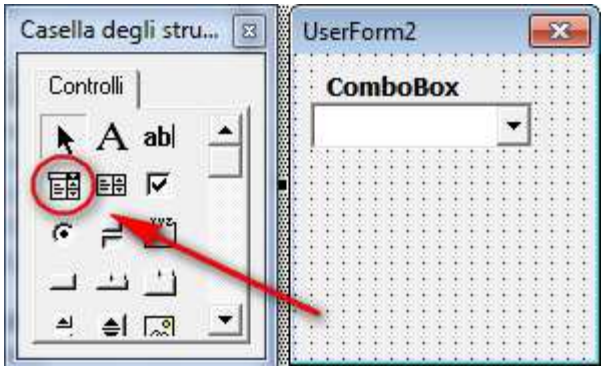


Fig. 1

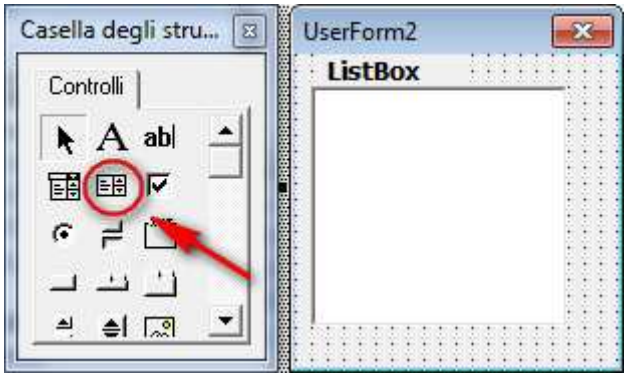


Fig. 2

Differenza tra ListBox e ComboBox

- In un ComboBox è visibile solo un elemento dell'elenco a discesa, e i valori della lista sono visibili utilizzando il menu a tendina, mentre un ListBox mostra un certo numero di valori con o senza una barra di scorrimento.
- In un ComboBox è possibile selezionare solo un'opzione dalla lista, mentre in un ListBox è possibile selezionare più opzioni dall'elenco.
- In un ComboBox è possibile inserire un valore digitandolo nell'area di testo se non è incluso nella lista, cosa che non è possibile fare in un ListBox.
- Il controllo CheckBox può essere utilizzato all'interno di un ListBox, ma non all'interno del ComboBox, inoltre il ListBox consente di visualizzare una casella di controllo accanto a ogni voce in elenco, per consentire all'utente di selezionare gli elementi. Per utilizzare il CheckBoxe in un controllo ListBox, si deve impostare la proprietà *ListStyle* nella finestra Proprietà su *fmListStyleOption* oppure utilizzando il codice VBA in questo modo:*ListBox1.ListStyle = fmListStyleOption*

Tutte le proprietà e i metodi indicati di seguito sono comuni a ListBox e ComboBox, salvo diversa indicazione, inoltre negli esempi di seguito riportati, i codici VBA devono essere inseriti nel modulo di codice del form, se non diversamente specificato.

Metodo AddItem:
Questo metodo permette di aggiunge una voce alla lista in un ListBox a colonna singola o in un ComboBox. La sintassi è la seguente: *Control.AddItem(Item, Index)*, che nel nostro caso diventa: *ListBox1.AddItem(Item, Index)*, dove *Item* specifica l'elemento o la riga da aggiungere e *Index* è un numero intero che specifica la posizione in cui il nuovo elemento è collocato all'interno della lista, se omesso, viene aggiunto l'elemento alla fine. I numeri di posizione o di riga iniziano con zero, e il primo elemento ha il numero 0, e così via. Il valore di indice non può essere maggiore del numero totale di righe. Il metodo*AddItem* può essere usato solo con un codice VBA. Per popolare un ComboBox è possibile utilizzare vari metodi, per esempio, se usiamo una casella combinata per permettere all'utente di vare una scelta "fissa" come può essere di scegliere una regione, un tipo di vettura etc. possiamo usare il metodo*AddItem* e impostare le varie voci direttamente dal codice, inoltre l'operazione di inserimento dati deve essere fatta nell'evento *Initialize* della Userfom in questo modo.

```
Private Sub UserForm_Initialize()
ComboBox1.Clear
ComboBox1.AddItem "Veneto"
ComboBox1.AddItem "Sardegna"
ComboBox1.AddItem "Lazio"
ComboBox1.ListIndex = 0
End Sub
```

Se mandiamo in esecuzione questo codice ci viene riportata questa finestra

Fig. 3

Da notare che la prima voce dell'enunciato è l'istruzione *ComboBox1.Clear* che serve per svuotare di eventuali dati rimasti in memoria del ComboBox in operazioni precedenti, per essere certi che la casella combinata è vuota, si può usare una condizione come questa : *If ComboBox1.ListCount >= 1 Then ComboBox1.Clear*

L'istruzione **ListCount** è una proprietà per determinare il numero di righe presenti in una casella combinata o di riepilogo, che inserendola in un enunciato IF verifica se nella casella sono presenti dei dati, infatti se *ListCount* è maggiore o uguale a 1 allora si esegue lo svuotamento del ComboBox tramite l'istruzione Clear, evitando così il rischio di trovarsi i dati ripetuti nella casella combinata. La penultima istruzione *ComboBox1.ListIndex = 0* Indica che si vuole far apparire la prima voce dell'elenco nel Combo come si vede in **figura 3**, se avessimo usato la dicitura *ComboBox.ListIndex = 1* avremmo come prima voce la seconda (Sardegna) presente nell'elenco. Possiamo utilizzare anche un ciclo per popolare il ComboBox in questo modo

```
Private Sub UserForm_Initialize()
 With ComboBox1
 .Clear
 .AddItem "Veneto"
 .AddItem "Sardegna"
 .AddItem "Lazio"
 .ListIndex =0
 End With
End Sub
```

Oppure se vogliamo popolare il ComboBox con i dati presenti in un Range del foglio di lavoro possiamo usare questo enunciato:

```
Private Sub UserForm_Initialize()
If ListBox1.ListCount >= 1 Then ListBox1.Clear
i = 1
Do Until Sheets("Foglio1").Cells(i, 1).Value = ""
With Sheets("Foglio1")
 elemento = .Cells(i, 1).Value
 End With
 ListBox1.AddItem elemento
 i = i + 1
 Loop
ListBox1.ListIndex = 0
End Sub
```

Per quanto riguarda il controllo ListBox non c'è nessuna differenza da quanto finora esposto, tutti i metodi e gli enunciati proposti possono essere usati anche per la casella di riepilogo ListBox avendo l'accortezza di sostituire il riferimento ComBox1 con ListBox1 (oppure il nome assegnato al controllo)

Metodo Clear

Questo metodo rimuove tutti gli elementi in un controllo ComboBox o ListBox e presenta la seguente Sintassi: *Control.Clear*, inoltre è da tenere presente che il metodo non funziona se ComboBox o ListBox viene associato a una fonte di dati utilizzando l'istruzione *RowSource*, in quanto i dati devono essere cancellati prima dell'uso del metodo Clear

Proprietà BoundColumn

Questa proprietà specifica la colonna da cui estrarre il valore da inserire in un controllo ComboBox o ListBox. La prima colonna ha un valore *BoundColumn* di 1, la seconda colonna ha un valore di 2, e così via, impostando il valore BoundColumn su 1 si assegnerà il valore della colonna 1 al ComboBox o ListBox permettendo così l'inserimento dei valori presenti nella colonna nel controllo. Impostando il valore di BoundColumn a 0 si assegna il valore della struttura*ListIndex*, che corrisponde al numero della riga selezionata, come valore del controllo ComboBox o ListBox. Questa impostazione è utile se si desidera determinare la riga della voce selezionata. La proprietà BoundColumn può essere impostata nella finestra Proprietà e può essere utilizzato anche con un codice VBA.

Esempio: Impostando il valore BoundColumn a 0 si assegna il valore della proprietà ListIndex come valore del controllo

```
Private Sub UserForm_Initialize()
With ListBox1
.ColumnHeads = True
.ColumnCount = 2
.ColumnWidths = "50;0"
.RowSource = "=Foglio2!A2:B6"
.MultiSelect = fmMultiSelectSingle
.TextColumn = 1
.BoundColumn = 0
End With
End Sub

Private Sub CommandButton1_Click()
If ListBox1.Value <> "" Then
TextBox1.Value = ListBox1.Value + 2
End If
End Sub
```

Proprietà Column

Si riferisce ad una colonna specifica, o combinazione di colonna e riga, in un ComboBox o ListBox a più colonne. Sintassi: Control.Column (iColumn, iRow). La proprietà *Column* può essere utilizzata solo con un codice VBA e non è disponibile in fase di progettazione. *iColumn* specifica il numero di colonna in cui la prima colonna della lista ha valore 0 e *iRow* specifica il numero di riga la prima riga della lista ha valore 0. Sia iColumn che iRow sono valori interi che vanno da 0 fino al numero di colonne e righe (rispettivamente) nell'elenco meno 1. Se si specifica entrambi i numeri di colonna e riga si farà riferimento ad un elemento specifico, mentre specificando solo il numero di colonna si farà riferimento a una colonna specifica. Per esempio [i]ListBox1.Column (1) è/i] si riferisce alla seconda colonna. È possibile copiare una matrice bidimensionale di valori in un controllo ListBox o ComboBox, utilizzando Column (o list) piuttosto che aggiungere ogni singolo elemento con il metodo AddItem.

Esempio: Popolare il ListBox utilizzando il metodo AddItem e List e utilizzare la proprietà Column per assegnare il contenuto di ListBox a TextBox

```
Private Sub UserForm_Initialize()
With ListBox1
.ColumnCount = 3
.ColumnWidths = "50;50;50"
.ColumnHeads = False
.RowSource = "=Foglio2!A2:B6"
.MultiSelect = fmMultiSelectMulti
End With
TextBox1 = ""
End Sub

Private Sub CommandButton1_Click()
'Il metodo AddItem non funziona se ListBox è associato ai dati, quindi la riga di origine
viene cancellata
ListBox1.RowSource = ""
'Crea una nuova riga con AddItem
ListBox1.AddItem "Banana"
'aggiunge un elemento nella seconda colonna della prima riga
ListBox1.List(0, 1) = "Martedì"
'aggiunta di elementi in 3 colonne della prima riga
ListBox1.List(0, 2) = "Giorno 2"
ListBox1.AddItem "Arancione"
'Aggiunge un elemento nella seconda colonna della seconda riga
ListBox1.Column(1, 1) = "Mercoledì"
'aggiunta di elementi in 3 colonne della seconda fila
ListBox1.Column(2, 1) = "Giorno 3"
ListBox1.AddItem "apple", 0
'Crea una nuova riga con AddItem e si posiziona nella riga 1
ListBox1.List(0, 1) = "Lunedì"
'Aggiunta di elementi in 3 colonne e posiziona questa riga come la prima
ListBox1.List(0, 2) = "Giorno 1"
'elemento in colonna 3 e numero di riga 2 del ListBox
TextBox1.Value = ListBox1.Column(2, 1)
End Sub
```

Proprietà ColumnCount

Specifica il numero di colonne da visualizzare in un controllo ComboBox o ListBox, un valore 0 di ColumnCount non visualizza nessuna colonna e questa proprietà può essere impostata nella finestra Proprietà oppure utilizzando un codice VBA.

Proprietà ColumnHeads

ColumnHeads restituisce un valore booleano (True o False) che determina la visualizzazione delle intestazioni di colonna per un ComboBox o ListBox e può essere impostata nella finestra Proprietà oppure utilizzando un codice VBA. Le intestazioni di colonna possono essere visualizzati solo se ColumnHeads è impostato su True nella finestra Proprietà oppure utilizzando il codice: *ListBox1.ColumnHeads = True*

Proprietà List

La proprietà *List* viene utilizzata in combinazione con *ListCount* e *ListIndex* per restituire gli elementi in un controllo ListBox o ComboBox. Sintassi :*Control.List (iRow, iCol)* . Ogni elemento in una lista ha un numero di riga e un numero di colonna, in cui i numeri di riga e di colonna iniziano da zero, iRow specifica il numero di riga e nel caso iRow = 2 rappresenta la terza riga della lista, mentre invece iColumn specifica il numero di colonna e nel caso che iColumn = 0 indica la prima colonna della lista, omettendo questo argomento iColumn recupererà la prima colonna. Si deve specificare iColumn solo per un ListBox o un ComboBox a più colonne. La proprietà List può essere utilizzata solo con un codice VBA e non è disponibile in fase di progettazione

Esempio: Utilizzare Selected, List per visualizzare e selezionare più elementi in un ListBox

```
Private Sub UserForm_Initialize()
With ListBox1
.ColumnHeads = True
.ColumnCount = 2
.ColumnWidths = "50;0"
.RowSource = "=Foglio3!A2:B6"
.MultiSelect = fmMultiSelectMulti
.TextColumn = 1
End With
With TextBox1
.MultiLine = True
.ControlSource = "=Foglio3!F2"
.Value = ""
End With
End Sub

Private Sub CommandButton1_Click()
TextBox1.Value = ""
For n = 0 To ListBox1.ListCount - 1
If ListBox1.Selected(n) = True Then
If TextBox1.Value = "" Then
TextBox1.Value = ListBox1.List(n, 1)
Else
TextBox1.Value = TextBox1.Value & vbCrLf & ListBox1.List(n, 1)
End If
End If
Next n
End Sub
```

Proprietà ListCount

Determina il numero totale di righe in un controllo ListBox o ComboBox, e questa proprietà può essere utilizzata solo con un codice VBA e non è disponibile in fase di progettazione.

Proprietà ListIndex

Determina quale elemento viene selezionato in un controllo ComboBox o ListBox. Il primo elemento di un elenco ha un valore di ListIndex uguale a 0, il secondo elemento ha un valorei 1, e così via, quindi, è un valore intero compreso tra 0 e il numero totale di elementi in un controllo ComboBox o ListBox meno 1. Questa proprietà può essere utilizzata solo con un codice VBA e non è disponibile in fase di progettazione. Nota: In una selezione multipla attivata in un ListBox, ListIndex restituisce l'indice della riga che ha il focus, a prescindere dal fatto che sia selezionata la riga o meno.

Proprietà ListRows

Specifica il numero massimo di righe che verranno visualizzate nella parte casella di riepilogo di un ComboBox. Il valore predefinito è 8. Nota: Se il numero effettivo di elementi della lista superi tale valore massimo della proprietà ListRows, apparirà una barra di scorrimento verticale nella casella di riepilogo del ComboBox e le voci in eccesso possono essere visualizzate scorrendo verso il basso). ListCount può essere utilizzata con la proprietà ListRows per specificare il numero di righe da visualizzare in un controllo ComboBox e può essere impostata nella finestra Proprietà oppure con un codice VBA. La proprietà ListRows è valida per ComboBox e non per ListBox.

Esempio: Utilizzare la proprietà ListCount e ListRows, per impostare il numero di righe da visualizzare in ComboBox

```
Private Sub UserForm_Initialize()
With ComboBox1
If .ListCount > 5 Then
.ListRows = 5
Else
.ListRows = .ListCount
End If
End With
End Sub
```

Proprietà MultiSelect

Specifica se sono consentite selezioni multiple e sono disponibili 3 impostazioni:
fmMultiSelectSingle (valore 0), è l'impostazione predefinita, in cui solo un singolo elemento può essere selezionato
fmMultiSelectMulti (valore 1), permette selezioni multiple in cui un elemento può essere selezionato o deselezionato facendo clic del mouse o premendo la barra spaziatrice
fmMultiSelectExtended (valore 2) permette selezioni multiple, in cui premendo il tasto SHIFT e contemporaneamente spostando la freccia su o giù (o premere MAIUSC e facendo clic del mouse) continua la selezione dalla voce precedentemente selezionata con la selezione corrente (cioè un continuo della selezione); questa opzione permette anche di selezionare o deselezionare una voce premendo CTRL e cliccando il mouse.

La proprietà *MultiSelect* può essere impostata nella finestra Proprietà oppure con un codice VBA. Nota: La proprietà MultiSelect è valida per ListBox e non per ComboBox . Quando vengono effettuate selezioni multiple, gli elementi selezionati possono essere determinati solo mediante la proprietà Selected immobili della ListBox. La struttura Selected avrà valori che vanno da 0 a ListCount meno 1 e sarà True se l'elemento è selezionato e False se non selezionata.

Esempio: Determinare l'elemento selezionato in un controllo ListBox a selezione singola

```
Private Sub CommandButton1_Click()
If ListBox1.Value <> "" Then
MsgBox ListBox1.Value
End If
End Sub
```

Metodo RemoveItem

Viene utilizzato per rimuovere una riga dalla lista in un ComboBox o ListBox.
Sintassi: *Control.RemoveItem (row_index)*, dove *Row_index* è il numero di riga che deve essere rimosso, considerando che la prima riga ha valore 0. Il metodo RemoveItem non funziona se ComboBox o ListBox viene associato ai dati, quindi utilizzando RowSource i dati devono essere cancellati prima dell'uso, inoltre questo metodo può essere utilizzato solo con un codice VBA.

Proprietà RowSource

Specifica la riga di origine di una lista (che potrebbe essere un intervallo del foglio di lavoro), per un ComboBox o ListBox. La proprietà RowSource può essere impostata nella finestra Proprietà oppure con un codice VBA. Per impostare RowSource nella finestra delle Proprietà, digitare senza virgolette il Range di origine in questo modo: " = *Foglio2 A2: A6*" che popola il ComboBox o il ListBox con i valori presenti nelle celle A2: A6 del Foglio2. Il codice VBA per questo passaggio è: *ListBox1.RowSource = "= Foglio2 A2: A6"*. Non è necessario utilizzare il segno di uguale ("= Foglio2 A2: A6"), impostando la proprietà con *ListBox1.RowSource = "Foglio2 A2: A6"* avrà lo stesso effetto.

Proprietà Selected

Specifica se un elemento viene selezionato in un controllo ListBox. Sintassi: *Control.Selected (Item_Index)* e restituisce Vero o Falso, se l'elemento è selezionato oppure no, impostando a True per selezionare la voce o rimuovere la selezione [vale a dire. Control.Selected (Item_Index) = True]. Item_Index è un valore intero compreso tra 0 e il numero di elementi nella lista meno 1, indicando la sua posizione relativa nella lista, vale a dire *ListBox.Selected (2) = True* seleziona il terzo elemento della lista. Questa proprietà è particolarmente utile quando si lavora con selezioni multiple e può essere utilizzata solo con un codice VBA e non è disponibile in fase di progettazione. Nota: In una selezione multipla in un ListBox, ListIndex restituisce l'indice della riga che ha il focus, a prescindere dal fatto che sia selezionata la riga o meno, da cui la proprietà Selected della ListBox può essere utilizzata per impostare una nuova selezione. In una selezione standard del ListBox, ListIndex restituisce l'indice dell'elemento selezionato e quindi dovrebbe essere usata per impostare una nuova selezione.

Esempio: Determinare gli elementi selezionati in una selezione multipla ListBox utilizzando selected e List

```
Private Sub CommandButton1_Click()
For n = 0 To ListBox1.ListCount - 1
If ListBox1.Selected(n) = True Then
MsgBox ListBox1.List(n)
End If
Next n
End Sub
```

Proprietà Style

Questa proprietà è valida per ComboBox e non per ListBox e determina se la scelta della voce può essere fatta dalla lista a discesa oppure digitando nel campo di testo il valore della ComboBox. Dispone di due impostazioni:

- *fmStyleDropDownCombo* (valore 0). L'utente ha entrambe le opzioni di digitare un valore personalizzato nell'area di testo o selezionare dall'elenco a discesa. Questo è il valore predefinito.
- *fmStyleDropDownList* (valore 2). L'utente può selezionare solo dall'elenco a discesa, come in ListBox.

La proprietà *Style* può essere impostata nella finestra Proprietà oppure con codice VBA.

Proprietà TextColumn

Specifica la colonna da visualizzare in un controllo ComboBox o ListBox a più colonne, quando una riga è selezionata dall'utente. La prima colonna ha un valore TextColumn di 1, la seconda ha un valore di 2, e così via. Impostare il valore di TextColumn a 0 visualizza il valore ListIndex (che è il numero della riga selezionata) nella proprietà TextColumn, questa impostazione è utile se si desidera determinare la riga della voce selezionata.
La proprietà *TextColumn* può essere impostata nella finestra Proprietà o con codice VBA. Nota: In un ComboBox, quando un utente seleziona un elemento, la colonna specificata nella proprietà TextColumn verrà visualizzato nella parte casella di testo del ComboBox.

Esempio: Utilizzo di TextColumn per visualizzare prima colonna e BoundColumn per la seconda colonna in una ListBox a selezione singola

```
Private Sub UserForm_Initialize()
With ListBox1
.ColumnHeads = True
.ColumnCount = 2
.ColumnWidths = "50;0"
.RowSource = "=Foglio2!A2:B6"
.MultiSelect = fmMultiSelectSingle
.TextColumn = 1
.BoundColumn = 2
End With
End Sub

Private Sub CommandButton1_Click()
If ListBox1.Value <> "" Then
TextBox1.Value = ListBox1.Value & " cms"
End If
End Sub
```

Aggiungere elementi per popolare un controllo ListBox o ComboBox

Impostare la proprietà RowSource di un controllo ListBox o ComboBox in un form utente per un elenco statico:
Me.ListBox1.RowSource = "Foglio1 A1: B6" oppure *Me.ListBox1.RowSource = "= Foglio1 A1: B6"*
Mentre per un elenco dinamico: *Me.ListBox1.RowSource = "Foglio1 A1: B" . & Foglio1.Cells (Rows.Count, "B") End (xlUp) Row*

Esempio: Popolare un ComboBox impostando la proprietà RowSource di una lista denominata

```
Private Sub UserForm_Initialize()
With ComboBox1
.ColumnCount = 2
.ColumnWidths = "50;50"
.ColumnHeads = True
.RowSource = "Foglio1!nome_intervallo"  <<<< da cambiare con Vs. nome intervallo
End With
End Sub
```

Popolare un ComboBox o ListBox da un array a colonna singola in un ListBox:
ListBox1.List = Array ("Riga1", "Riga2", "Riga3", "Riga4")

in un ComboBox:
ComboBox1.List = array ("Riga1", " Riga2", " Riga3", " Riga4")

Compilare un ListBox da una matrice denominata myArray:
Dim myArray As Variant
myArray = Array ("Adidas", "Nike", "Reebok")
Me.ListBox1.List = myArray

Popolare una singola colonna in un ComboBox:
Dim i As Integer
Dim myArray As Variant
myArray = Array ("Adidas", "Nike", "Reebok", "Puma", "Polo")
For i = LBound(myArray) To UBound(myArray)
Me.ComboBox1.AddItem myArray(i)
Next

Esempio: Popolare un ListBox a più colonne da un Range del foglio di lavoro

```
Private Sub UserForm_Initialize()
With ListBox1
.ColumnCount = 3
.ColumnWidths = "50;50;50"
.ColumnHeads = False
End With
Dim rng As Range
Set rng = Foglio1.Range("A1:C6")
Me.ListBox1.List = rng.Cells.Value
End Sub
```

Esempio: Popolare un ListBox a più colonne dal Range del Foglio di lavoro

```
Private Sub UserForm_Initialize()
With ListBox1
.ColumnCount = 3
.ColumnWidths = "50;50;50"
.ColumnHeads = False
End With
Dim var As Variant
var = Foglio1.Range("A1:C6")
Me.ListBox1.List = var
End Sub
```

Esempio: Popolare un ListBox a più colonne dal Range di un foglio di lavoro dopo aver posizionato i dati di in una matrice a 2 dimensioni

Codice:

```
Option Base 1
Private Sub UserForm_Initialize()
Dim rng As Range
Dim cell As Range
Dim totalRiga As Integer, totalColon As Integer
Dim iRow As Integer, iCol As Integer
Dim myArray() As Variant
Set rng = Foglio1.Range("A1:C6")
totalRiga = Foglio1.Range("A1:C6").Rows.Count
totalColon = Foglio1.Range("A1:C6").Columns.Count
ReDim myArray(totalRiga, totalColon)
For Each cell In rng
For iRow = 1 To totalRiga
For iCol = 1 To totalColon
myArray(iRow, iCol) = rng.Cells(iRow, iCol)
Next iCol
Next iRow
Next
With ListBox1
.ColumnCount = 3
.ColumnWidths = "50;50;50"
.ColumnHeads = False
.List = myArray
End With
End Sub
```

Esempio: Caricare una matrice a 2 dimensioni in un ListBox utilizzando la proprietà List

Codice:

```
Private Sub UserForm_Initialize()
With ListBox1
.ColumnCount = 3
.ColumnWidths = "50;50;50"
.ColumnHeads = False
End With
With ListBox2
.ColumnCount = 3
.ColumnWidths = "50;50;50"
.ColumnHeads = False
End With
End Sub
```

```
Private Sub CommandButton1_Click()
Dim myArray(3, 3)
For n = 0 To 2
myArray(n, 0) = n + 1
Next n
myyArray(0, 1) = "R1C2"
myArray(1, 1) = "R2C2"
myArray(2, 1) = "R3C2"
myArray(0, 2) = "R1C3"
myArray(1, 2) = "R2C3"
myArray(2, 2) = "R3C3"
ListBox1.List() = myArray
ListBox2.Column() = myArray
End Sub
```

Esempio: Popolare un ComboBox con i 12 mesi in un anno

Codice:

```
Private Sub UserForm_Initialize()
With ComboBox1
.ColumnCount = 1
.ColumnWidths = "50"
.ColumnHeads = False
.RowSource = ""
End With
For n = 1 To 12
ComboBox1.AddItem Format(DateSerial(2014, n, 1), "mmmm")
Next n
End Sub
```

Esempio: Popolare una ListBox a più colonne da un Range utilizzando il metodo AddItem e List

Codice:

```
Private Sub UserForm_Initialize()
With ListBox1
.ColumnCount = 3
.ColumnWidths = "50;50;50"
.RowSource = ""
End With
End Sub

Private Sub CommandButton1_Click()
Dim conta As Long
Dim totalRiga As Long
totalRiga = Foglio4.Cells(Rows.Count, "A").End(xlUp).Row
conta = 0
Do
With Me.ListBox1
conta = conta + 1
.AddItem Foglio4.Cells(conta, 1).Value
.List(.ListCount - 1, 1) = Foglio4.Cells(conta, 1).Offset(0, 1).Value
.List(.ListCount - 1, 2) = Foglio4.Cells(conta, 1).Offset(0, 2).Value
End With
Loop Until conta = totalRiga
End Sub
```

```
Dim rng1 As Range, rng2 As Range, rngUnion As Range
'impostare un blocco contiguo di celle come primo intervallo (o Range)
Set rng1 = Range ("A1: B2")
'impostare un altro blocco contiguo di celle come secondo intervallo
Set rng2 = Range ("D3: E4")
'assegnare una variabile (oggetto range) per rappresentare l'unione dei due intervalli
Set rngUnion = Union (rng1, rng2)
'colore interno impostato per l'intervallo che è l'unione di 2 oggetti Range
rngUnion.Interior.Color = vbYellow
```

I Controlli ScrollBar e SpinButton

Ogni volta che l'utente ha un discreto numero di scelte da fare per l'inserimento dei dati in un foglio di lavoro di Excel è possibile risparmiare tempo automatizzando il modo in cui si inseriscono questi dati. È possibile farlo in diversi modi e uno di loro è quello di utilizzare un pulsante di selezione o una barra di scorrimento.

Il Controllo ScrollBar

Un controllo *ScrollBar*, o barra di scorrimento, permette di cambiare incrementando o decrementando il valore visualizzato da altri controlli come UserForm, TextBox, Label, etc. o il valore in un intervallo di celle quando un utente fa clic sulle frecce di scorrimento, oppure trascina la casella di scorrimento o clicca in una zona compresa tra una freccia di scorrimento e la casella di scorrimento. Il componente ScrollBar fornisce all'utente uno strumento per gestire un valore definito dal programma e può essere rappresentato graficamente in verticale o orizzontale e al momento della sua creazione VBE lo orienta verticalmente per default, ma possono essere ridimensionati in modo da estendersi orizzontalmente e assumere tale orientamento.

Le principali proprietà di questo controllo sono:

Value: Restituisce o imposta il valore del componente utilizzando dati di tipo Long
SmallChange: Specifica il cambiamento incrementale, come un valore intero (variabile Long), che si verifica quando un utente fa clic sulla freccia di scorrimento. Il valore di default è 1.
LargeChange: Specifica il cambiamento incrementale quando l'utente fa clic tra una freccia di scorrimento e la casella di scorrimento. Il valore di default è 1.
Min e Max: Sono valori interi (Long) che specificano il valore massimo e minimo accettabili del controllo ScrollBar.

In una barra di scorrimento verticale cliccando la freccia di scorrimento verso il basso, aumenta il valore e la posizione più bassa visualizza il valore massimo (sarà inverso quando si sceglie la freccia di scorrimento). In una barra di scorrimento orizzontale clic sulla freccia di scorrimento a destra aumenta il valore e la posizione più a destra visualizza il valore massimo (sarà inverso quando si fa clic sulla freccia di scorrimento a sinistra).
Orientation: Determina una barra di scorrimento verticale o una barra di scorrimento orizzontale. Dispone di 3 impostazioni:

- *fmOrientationAuto* (Valore -1) - Questo è il valore predefinito in cui le dimensioni della ScrollBar sono determinare automaticamente, sia in verticale o orizzontale, nei casi in cui il valore della larghezza è più alto dell'altezza, allora la ScrollBar è orizzontale e dove l'altezza è più alta della larghezza, allora la ScrollBar è verticale
- *FmOrientationVertical* (Valore 0) – La ScrollBar è verticale
- *FmOrientationHorizontal* (valore 1) – La ScrollBar è orizzontale.

Esempio: Calcolare la rata di un mutuo, utilizzando i controlli ScrollBar

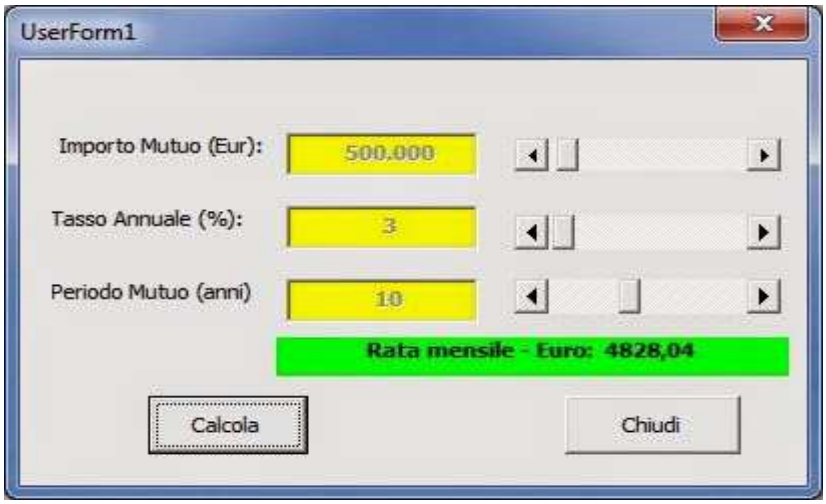


Fig. 1

```
Private Sub ScrollBar1_Change()
'facendo clic sulla freccia di scorrimento si incrementerà importo di 5.000 euro e facendo clic
'tra una freccia di scorrimento e la casella di scorrimento si incrementerà l'importo da 100.000
TextBox1.Value = ScrollBar1.Value * 1000
TextBox1.Value = Format(TextBox1.Value, "#,##0")
End Sub

Private Sub ScrollBar2_Change()
'facendo clic sulla freccia di scorrimento si incrementerà il tasso dello 0,1% e cliccando tra una
freccia di 'scorrimento e la casella di scorrimento si incrementerà il tasso dell'1%
TextBox2.Value = ScrollBar2.Value / 10
End Sub

Private Sub ScrollBar3_Change()
'clic sulla freccia si incrementerà il periodo di 0,5 anni e cliccando tra una 'freccia di scorrimento e
la casella di scorrimento si incrementa di 2 anni
TextBox3.Value = ScrollBar3.Value / 2
End Sub

Private Sub CommandButton1_Click()
'calcola la rata mensile utilizzando la funzione PMT
Dim mi As Currency
 If Not TextBox1.Value > 0 Then
MsgBox "Inserire Importo prestito!"
Exit Sub
ElseIf Not TextBox2.Value > 0 Then
MsgBox "Inserire Tasso di interesse annuo!"
Exit Sub
ElseIf Not TextBox3.Value > 0 Then
MsgBox "Inserire periodo Prestito!"
Exit Sub
Else
mi = Pmt((TextBox2.Value / 100) / 12, TextBox3.Value * 12, TextBox1.Value)
'Etichetta visualizza la rata mensile, arrotondata a 2 punti decimali
Label4.Caption = "Rata mensile - Euro:  " & Round(mi, 2) * -1
End If
End Sub
```

```
Private Sub CommandButton2_Click()
`pulsante di chiusura
Unload Me
End Sub
Private Sub UserForm_Initialize()
'Impostare le proprietà dei controlli della form
TextBox1.BackColor = RGB(255, 255, 0)
TextBox1.TextAlign = fmTextAlignCenter
TextBox1.Font.Bold = True
TextBox1.Enabled = False
Label1.Caption = "Importo Mutuo (Eur):"
Label1.TextAlign = fmTextAlignLeft
ScrollBar1.Min = 0
ScrollBar1.Max = 10000
ScrollBar1.Orientation = fmOrientationHorizontal
ScrollBar1.SmallChange = 5
ScrollBar1.LargeChange = 100
ScrollBar1.Value = 0
'Impostare le proprietà per i controlli del tasso di interesse annuo
TextBox2.BackColor = RGB(255, 255, 0)
TextBox2.TextAlign = fmTextAlignCenter
TextBox2.Font.Bold = True
TextBox2.Enabled = False
Label2.Caption = "Tasso Annuale (%):"
Label2.TextAlign = fmTextAlignLeft
ScrollBar2.Min = 0
ScrollBar2.Max = 1000
ScrollBar2.Orientation = fmOrientationHorizontal
ScrollBar2.SmallChange = 1
ScrollBar2.LargeChange = 10
ScrollBar2.Value = 0
'Impostare le proprietà per i controlli periodo del prestito
TextBox3.BackColor = RGB(255, 255, 0)
TextBox3.TextAlign = fmTextAlignCenter
TextBox3.Font.Bold = True
TextBox3.Enabled = False
Label3.Caption = "Periodo Mutuo (anni)"
Label3.TextAlign = fmTextAlignLeft
ScrollBar3.Min = 0
ScrollBar3.Max = 50
ScrollBar3.Orientation = fmOrientationHorizontal
ScrollBar3.SmallChange = 1
ScrollBar3.LargeChange = 4
ScrollBar3.Value = 0
'Impostare le proprietà per Label che visualizza Rata mensile
Label4.Caption = "Rata mensile: Eur "
Label4.TextAlign = fmTextAlignCenter
Label4.BackColor = RGB(0, 255, 0)
Label4.Font.Bold = True
End Sub
```

Il Controllo SpinButton

Un controllo *SpinButton*, o pulsante di selezione, viene spesso utilizzato per incrementare o decrementare il valore di un altro controllo, ad esempio un controllo Label e la proprietà *SmallChange* determina il valore di un controllo SpinButton quando cambia. Un controllo SpinButton è molto simile a un controllo ScrollBar e viene utilizzato per incrementare o diminuire il valore (cioè un numero, data, ora, etc.) visualizzato da altri controlli UserForm, TextBox, Label, etc. o il valore in un intervallo di celle. Un controllo SpinButton (noto anche come un controllo di selezione) funziona come un controllo ScrollBar, con proprietà simili (cioè SmallChange, Min, Max) e la proprietà SmallChange specifica il cambiamento incrementale, come un valore intero, che si verifica quando un utente fa clic sulla freccia di scorrimento. Un controllo SpinButton non ha una struttura *LargeChange* come in una ScrollBar e in una barra di scorrimento verticale cliccando la freccia di scorrimento diminuisce il valore mentre premendo la freccia di scorrimento su un Spinner verticale aumenta il valore.

La differenza tra i controlli ScrollBar e SpinButton è che la struttura ScrollBar può essere trascinata per modificare il valore del controllo su incrementi maggiori (pur mantenendo l'incremento basso per click), che i vantaggi di una ScrollBar per effettuare una selezione da tutto un numero di valori e coprono una gamma estremamente vasta.

Esempio: Utilizzare un controllo SpinButton per cambiare le date in una TextBox, all'interno di un intervallo specificato:

Fig. 2

```
Private Sub UserForm_Initialize()
'popolare il TextBox con una data
Dim dt As Date
'Non consentire l'immissione manuale in TextBox
TextBox1.Enabled = False
dt = "10/10/2014"
TextBox1.Text = dt
End Sub

Private Sub SpinButton1_SpinUp()
'aumentare di un giorno alla volta, quando siamo nello stesso mese
Dim dt_su As Date
Dt_su = "30/10/2014"
 If DateValue(TextBox1.Text) < dt_su Then
TextBox1.Text = DateValue(TextBox1.Text) + 1
End If
End Sub

Private Sub SpinButton1_SpinDown()
 'diminuire di un giorno alla volta, quando siamo nello stesso mese
Dim dt_giu As Date
Dt_giu = "01/10/2014"
If DateValue(TextBox1.Text) > dt_giu Then
TextBox1.Text = DateValue(TextBox1.Text) - 1
End If
End Sub
```

Esempio: Spostare le voci in alto e in basso selezionate in un ListBox utilizzando il controllo SpinButton

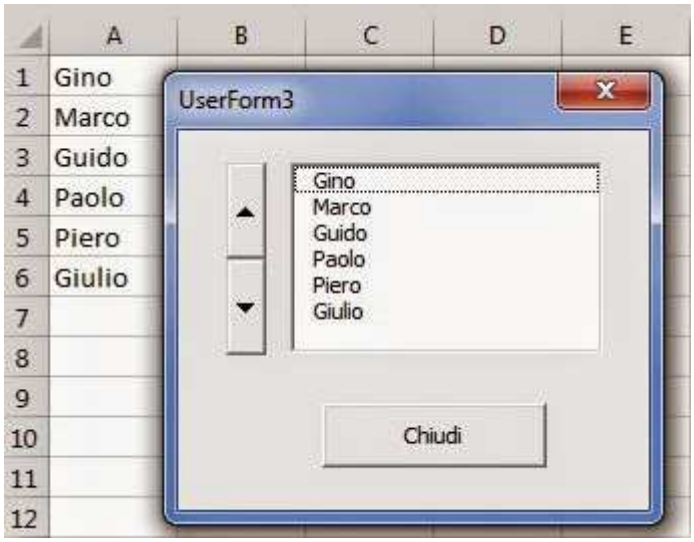


Fig. 3

```
Private Sub carica_Box()
Dim n As Integer, cell As Range, rng As Range
Set rng = Foglio1.Range("A1:A6")
For n = 1 To ListBox1.ListCount
ListBox1.RemoveItem ListBox1.ListCount - 1
Next n
For Each cell In rng.Cells
Me.ListBox1.AddItem cell.Value
Next cell
End Sub

Private Sub UserForm_Initialize()
carica_Box
End Sub

Private Sub SpinButton1_SpinUp()
'cliccare sulla freccia in alto per spostare l'elemento selezionato nel ListBox verso l'alto, sia nel
'ListBox che nel foglio di lavoro collegato
Dim n As Long
n = ListBox1.ListIndex
If n > 0 Then
Foglio1.Range("A" & n + 1).Value = Foglio1.Range("A" & n).Value
Foglio1.Range("A" & n).Value = ListBox1.Value
carica_Box
ListBox1.Selected(n - 1) = True
ElseIf ListBox1.ListIndex = 0 Then
MsgBox "Il primo elemento non può essere spostato in alto!"
Else
MsgBox "Seleziona una voce!"
End If
End Sub

Private Sub SpinButton1_SpinDown()
'cliccare sulla freccia in basso per spostare l'elemento selezionato nel ListBox verso il basso, sia
'nel ListBox che nel foglio di lavoro collegato
Dim n As Long
n = ListBox1.ListIndex
If n >= 0 And n < ListBox1.ListCount - 1 Then
Foglio1.Range("A" & n + 1).Value = Foglio1.Range("A" & n + 2).Value
Foglio1.Range("A" & n + 2).Value = ListBox1.Value
carica_Box
ListBox1.Selected(n + 1) = True
ElseIf ListBox1.ListIndex = ListBox1.ListCount - 1 Then
MsgBox "L'ultimo elemento non può essere spostato in basso!"
Else
MsgBox "Seleziona una voce!"
End If
End Sub

Private Sub CommandButton1_Click()
Unload Me
End Sub
```

Il controllo Image e RafEdit

Un controllo **Image** consente di visualizzare delle immagini nella propria applicazione. Per assegnare una foto a questo controllo in fase di progettazione, si utilizza la proprietà Image, inoltre è possibile assegnare un'immagine al controllo Image in fase di esecuzione utilizzando la funzione *LoadPicture*. Un aspetto importante è quello di inserire un'immagine nel controllo Image o adattare le dimensioni del controllo all'immagine, che determina come sarà visualizzata. Questo può essere fatto usando la proprietà *PictureSizeMode* come descritto di seguito.

La Proprietà ***PictureSizeMode*** specifica come viene visualizzata l'immagine, vale a dire, se il controllo Image non ha le stesse dimensione dell'immagine è possibile modificare la modalità di visualizzazione con le proprietà disponibili che sono:

- *fmPictureSizeModeClip* (valore 0) - se il controllo immagine non ha le stesse dimensioni dell'immagine ritaglia le parti dell'immagine che non rientrano nello spazio del controllo immagine.
- *fmPictureSizeModeStretch* (valore 1) – ridimensiona l'immagine per adattarla alcontrollo Image. Se le proporzioni dell'immagine e del controllo sono diverse l'immagine viene distorta
- *fmPictureSizeModeZoom* (valore 3) – Ridimensiona l'immagine senza distorcela e l'immagine viene ingrandita o rimpicciolita fino a che non raggiunge i margini orizzontali o verticali del controllo. Se il limite orizzontale viene raggiunto per primo, il rimanente spazio verticale apparirà vuoto e se il limite verticale viene raggiunto per primo, il rimanente spazio orizzontale apparirà vuoto.

La Proprietà PictureSizeMode può essere impostata nella finestra Proprietà oppure con codice VBA.

La Proprietà ***PictureAlignment*** specifica la posizione relativa o l'allineamento del quadro all'interno del controllo Image. Sono disponibili 5 impostazioni:

- *fmPictureAlignmentTopLeft* (valore 0)
- *fmPictureAlignmentTopRight* (valore 1)
- *fmPictureAlignmentCenter* (valore 2)
- *fmPictureAlignmentBottomLeft* (valore 3)
- *fmPictureAlignmentBottomRight* (valore 4).

Le impostazioni sono auto-esplicative, per esempio *fmPictureAlignmentBottomLeft* significa che l'immagine si allinea nell'angolo in basso a sinistra del controllo Image.

NB *Se l'immagine è impostata su fmSizeModeStretch nella proprietà PictureSizeMode, l'immagine si estende per adattarsi al suo controllo, quindi PictureAlignment non dispone di alcun effetto.*

Utilizzando la Funzione **LoadPicture** si assegna un'immagine al controllo Image in fase di esecuzione: si usa la seguente Sintassi: *ImageControl.Picture = LoadPicture (percorso)*

Esempio: Selezionare la foto da visualizzare nel controllo Image dal ListBox1

```
Private Sub UserForm_Initialize ()
ListBox1.AddItem "Immagine1"
ListBox1.AddItem "Immagine2"
ListBox1.AddItem "Immagine3"
ListBox1.AddItem "Immagine4”
End Sub

Private Sub ListBox1_Click ()
If ListBox1.Value = "immagine 1" Then
Image1.Picture = LoadPicture ("C:\Test\Immagini\immagine1.jpg")
ElseIf ListBox1.Value = "immagine2" Then
Image1.Picture = LoadPicture ("C:\Test\Immagini\immagine2.jpg")
ElseIf ListBox1.Value = "immagine3" Then
Image1.Picture = LoadPicture ("C:\Test\Immagini\immagine3.jpg")
ElseIf ListBox1.Value = "immagine4" Then
Image1.Picture = LoadPicture ("C:\Test\Immagini\immagine4.jpg")
End If
End Sub
```

Fig. 1

Esempio: Selezionare il percorso completo di un'immagine da visualizzare nel controllo Image dal ListBox1

```
Private Sub UserForm_Initialize ()
Dim img1 As String
Dim img2 As String
Dim img3 As String
Dim img4 As String

img1 = "C:\Test\Immagini\immagine1.jpg"
img2 = "C:\Test\Immagini\immagine2.jpg"
img3 = "C:\Test\Immagini\immagine3.jpg"
img4 = "C:\Test\Immagini\immagine4.jpg"

ListBox1.AddItem img1
ListBox1.AddItem img2
ListBox1.AddItem img3
ListBox1.AddItem img4
End Sub

Private Sub ListBox1_Click ()
Image1.Picture = LoadPicture (ListBox1.List (ListBox1.ListIndex))
End Sub
```

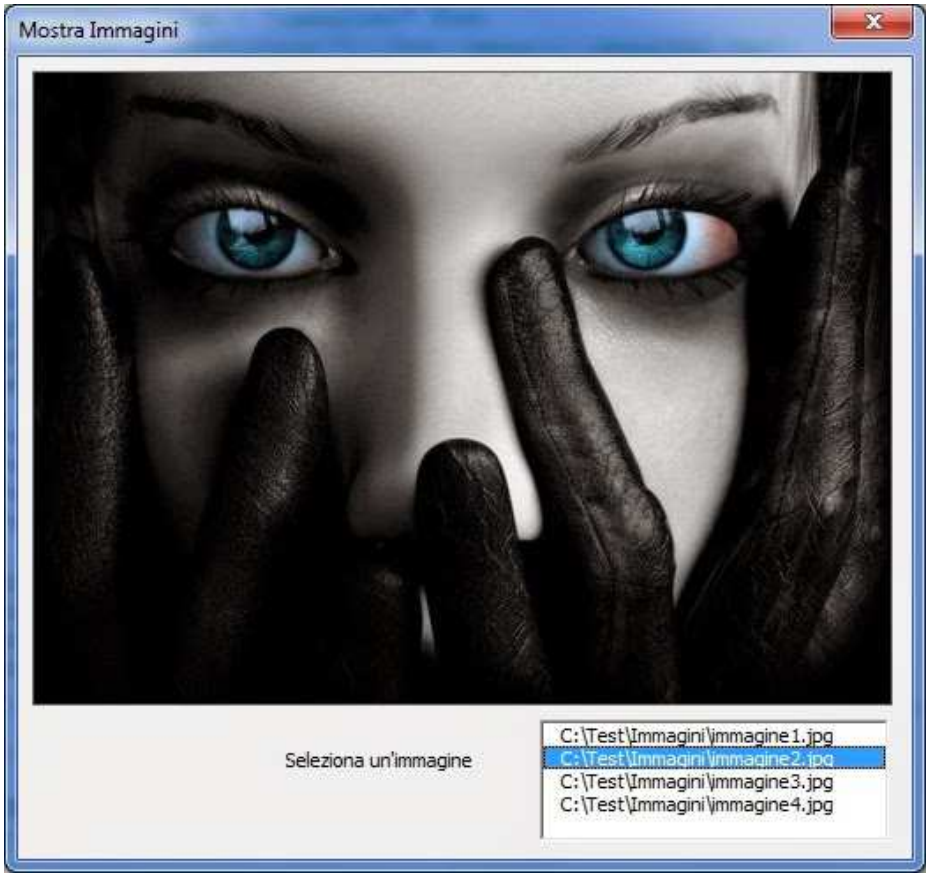


Fig. 2

Esempio: Visualizzare un'immagine nel pulsante di comando, all'attivazione della UserForm:

```
Private Sub UserForm_Initialize ()
CommandButton1.Picture = LoadPicture ("C:\Test\Immagini\img1.jpg")
End Sub
```

Fig. 3

Esempio: Visualizzare un immagine nel controllo Image cliccando sul pulsante di comando:

```
Private Sub CommandButton1_Click ()
With UserForm2
Image1.Picture = LoadPicture ("C:\Test\Immagini\immagine4.jpg")
End With
End Sub
```

Fig. 4

Metodo GetOpenFilename

Questo metodo consente all'utente di immettere un nome di file nella finestra di dialogo standard *Apri File*. Il metodo non apre qualsiasi file, viene visualizzata solo la casella per accettare il nome del file, e restituisce il nome inserito dall'utente o il nome del file selezionato, che potrebbe anche essere un percorso (cioè cartella e nome del file). La sintassi è la seguente:

Espressione GetOpenFilename ([FileFilter], [FilterIndex], [Titolo], [ButtonText], [Selezione multipla])

FileFilter è una stringa che specifica i criteri per filtrare i file. Si compone di coppie di stringhe e ogni parte comprende il carattere jolly ed è separato da virgole. Esempio: *FileFilter: = "File di testo (txt.), * txt, TIFF (* tif.), * tif, JPEG Files (* jpg.), * jpg"*

L'esempio seguente di FileFilter specifica 3 criteri di filtri di file (file di testo txt, i file TIFF e file JPEG) e consente all'utente di inserire o selezionare i file con estensione txt, Jpg e tif. Per includere più espressioni in un unico criterio di filtro di file, utilizzare punti e virgola per separare ogni espressione.
Esempio: *FileFilter: = "JPEG File Interchange Format (* jpg, * jpeg, * jfif; * GPE), * jpg, * jpeg, * jfif; * GPE"*

E' possibile utilizzare FileFilter specificando 1 criterio solo di filtro per i file (JPEG) e comprende quattro espressioni che consentono la selezione di file con estensione jpg, jpeg, jfif e GPE. Tralasciando questo argomento sarà di default il filtro "Tutti i file (*. *), *. *".

FilterIndex è il criterio di default del filtro dei file specificati dal numero indice che va da 1 al numero di criteri specifici di filters. Omettendo di menzionare FilterIndex, o specificando un numero di indice oltre la sua dotazione, verrà impostato il primo criterio del filtro file.

Titolo specifica il titolo della finestra di dialogo che viene visualizzata utilizzando il metodo *GetOpenFilename*. Il titolo predefinito è "Open" se non specificato.

MultiSelect se impostato a True consente la selezione multipla dei file e se impostato a False, che è anche il valore di default, permette la selezione di un singolo file.

Esempio: Usare LoadPicture con il metodo GetOpenFilename per caricare foto in un controllo Image.

```
Private Sub UserForm_Initialize()
Me.Image1.BackColor = RGB(152, 255, 152)
Me.Label1.Caption = "Clicca sulla casella verde per selezionare e caricare un'Immagine"
Me.CommandButton1.Caption = "Cliccare per Uscire"
End Sub

Private Sub Image1_Click()
Dim fileName As String
fileName = Application.GetOpenFilename(filefilter:="Tiff Files(*.tif;*.tiff),*.tif;*.tiff,JPEG
Files (*.jpg;*.jpeg;*.jfif;*.jpe),*.jpg;*.jpeg;*.jfif;*.jpe,Bitmap Files(*.bmp),*.bmp",
FilterIndex:=2, Title:="Seleziona un file", MultiSelect:=False)
If fileName = "False" Then
MsgBox "File non selezionato!"
Else
Me.Image1.Picture = LoadPicture(fileName)
Me.Repaint
Me.Label1.Caption = "Immagine Caricata"
End If
End Sub
```

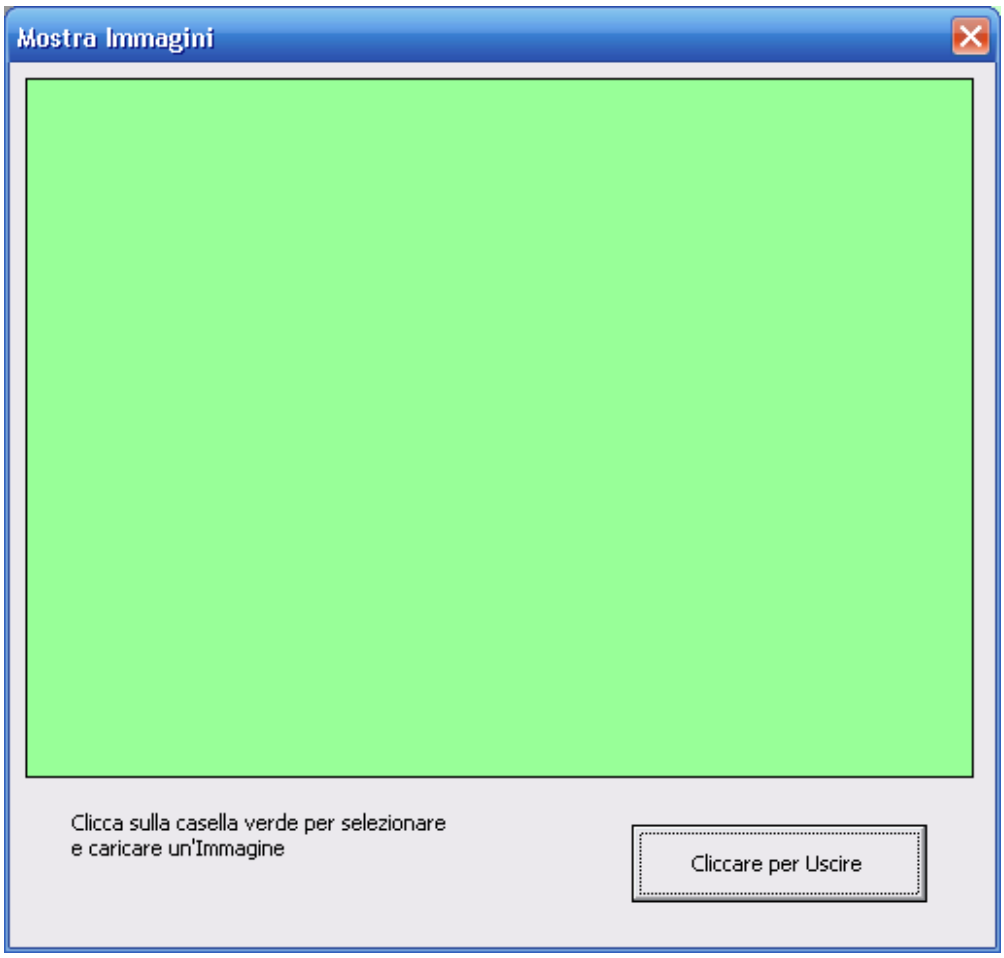

Mostra Immagini
Clicca sulla casella verde per selezionare
e caricare un'Immagine
Cliccare per Uscire

Fig. 5

Mostra Immagini
Seleziona un file
Cerca in:
Immagini
Strumenti
Documenti recenti
Desktop
Documenti
Risorse del computer
Risorse di rete
11
22
33
Nome file:
Tipo file:
JPEG Files
Apri
Annulla

Fig. 6

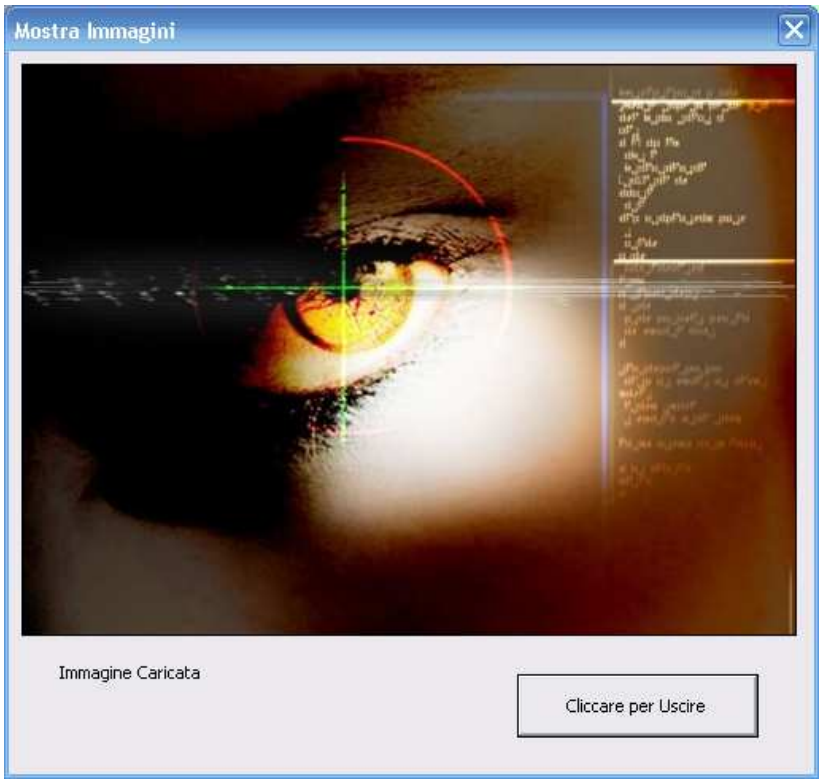


Fig. 7

Esempio: Usare LoadPicture con il metodo GetOpenFilename per caricare foto in un controllo Image

```
Private Sub UserForm_Initialize()
Me.Image1.BackColor = RGB(152, 255, 152)
Me.Label1.Caption = "Clicca sulla casella verde per selezionare e caricare un'Immagine"
Me.CommandButton1.Caption = "Cliccare per Uscire"
End Sub
Private Sub Image1_Click()
Dim fltr As String, ttl As String, fileName As String
Dim fltrIndx As Integer
Dim mltiSlct As Boolean
fltr = "Tiff Files(*.tif;*.tiff),*.tif;*.tiff,JPEG Files
(*.jpg;*.jpeg;*.jfif;*.jpe),*.jpg;*.jpeg;*.jfif;*.jpe,Bitmap Files(*.bmp),*.bmp"
fltrIndx = 2
ttl = "Seleziona un file"
mltiSlct = False
ChDrive "C"
ChDir "C:\Test\Immagini"
fileName = Application.GetOpenFilename(fltr, fltrIndx, ttl, , mltiSlct)
If fileName <> "False" Then
Me.Image1.Picture = LoadPicture(fileName)
Me.Repaint
Me.Label1.Caption = "Immagine Caricata"
End If
End Sub

Private Sub CommandButton1_Click()
Unload Me
End SubPrivate Sub CommandButton1_Click()
Unload Me
End Sub
```

Il risultato ottenuto è lo stesso visibile nelle Figure 5-6-7.

Il Controllo RefEdit

Il controllo *RefEdit* è disponibile solo in una UserForm e consente all'utente di selezionare una cella o un intervallo di celle del foglio di lavoro, permette anche di digitare direttamente nella casella di testo i riferimenti di cella. Nel campo del controllo RefEdit è visibile l'indirizzo della cella o l'intervallo di celle , che un utente seleziona o digita nel campo stesso. Per ottenere l'indirizzo di una cella o l'intervallo di celle memorizzate in un controllo RefEdit si utilizza la proprietà Value.

Esempio: Selezionare una cella nel foglio ed eseguire il codice premendo un pulsante

```
Private Sub CommandButton1_Click ()
Dim rngAddress As String
rngAddress = RefEdit1.Value
Range (rngAddress). Value = "Ciao"
Range (rngAddress). Interior.Color = RGB (255, 0, 0)
If MsgBox ("Azione eseguita. Vuoi Uscire?", VbQuestion + vbYesNo) = vbYes Then
Unload Me
End If
End Sub
```

Fig. 8

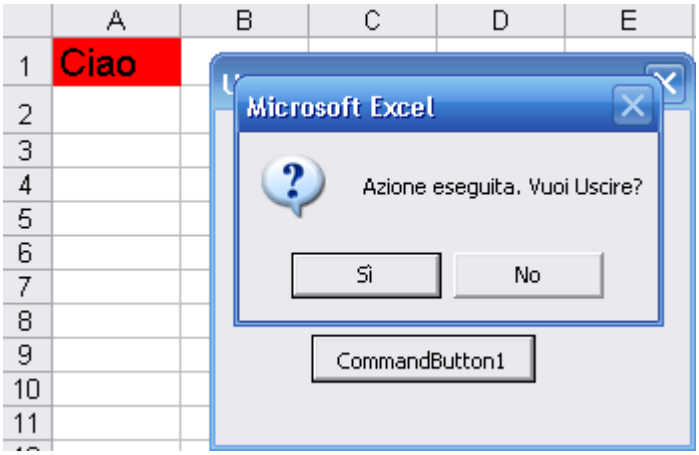


Fig. 9

Il Debug in Visual Basic Editor

Il **Debug** è un aspetto molto importante in programmazione e gli sviluppatori hanno bisogno di identificare e rettificare rapidamente gli errori durante la scrittura del codice. VBA fornisce numerosi strumenti di debug per risolvere i problemi nella fase di sviluppo, e oltre alla possibilità di aggiungere le routine di gestione degli errori, fornisce soluzioni rapide quando si manifestano gli errori durante l'esecuzione.

Indentazione del codice

Per poter facilitare la lettura e la comprensione del codice è buona cosa ricorrere all'indentazione del codice che consiste nel precedere le righe di codice con un certo numero di spazi e ha lo scopo di evidenziare i blocchi di codice per permettere di cogliere visivamente la struttura del programma, inoltre si deve indentare il codice mentre lo si sviluppa, non successivamente per renderlo "bello". Il numero di spazi è sempre multiplo di un certo valore scelto come base (in genere 3 o 4) e normalmente si può definire il tasto **Tab** tramite il menu **Strumenti – Opzioni** dell'Editor di VBE, in modo che introduca quel numero di spazi

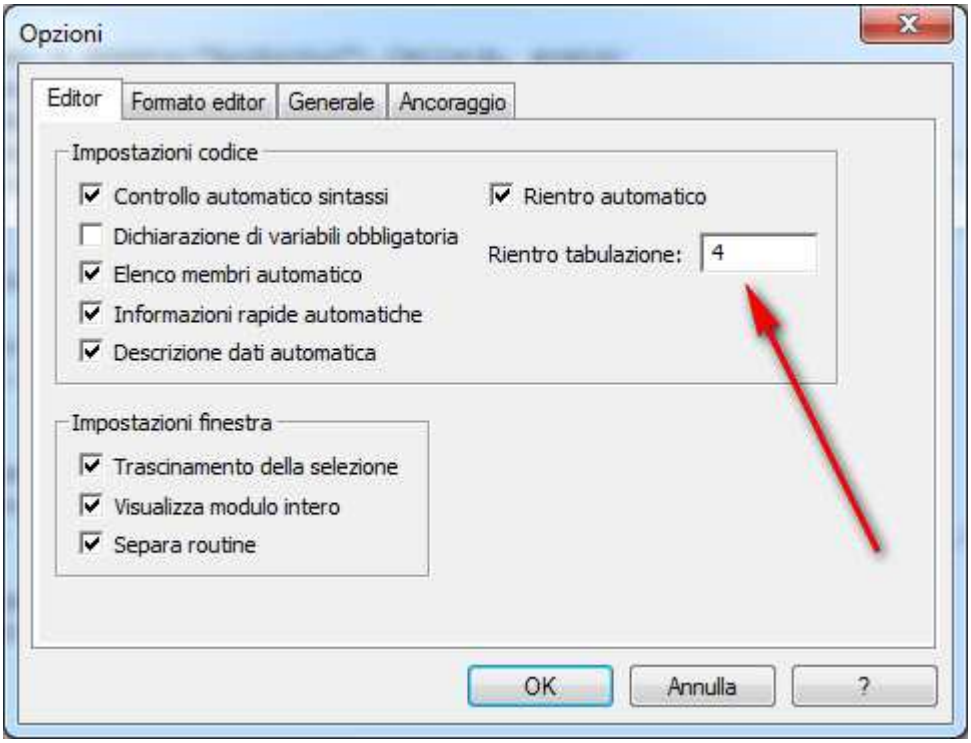


Fig. 1

Indentare il codice è un requisito non obbligatorio e viene spesso messo in secondo piano, tuttavia, è fondamentale per incrementare la leggibilità del codice, in particolare delle strutture di controllo come le condizioni o i loop allo scopo di separare più chiaramente le istruzioni e, in particolare, di rappresentare esplicitamente le relazioni di annidamento e viene considerata come una norma fondamentale di buona programmazione.

Esempio codice NON Indentato

```
Private Sub UserForm_Activate()
Dim elemento As String
If ListBox1.ListCount >= 1 Then ListBox1.Clear
i = 1
Do Until Sheets("Archivio").Cells(1, i).Value = Empty
 With Sheets("Archivio")
  elemento = .Cells(1, i).Value
 End With
  ListBox1.AddItem elemento
  i = i + 8
Loop
ListBox1.ListIndex = 0
svuota_box
End Sub
```

Esempio codice Indentato

```
Private Sub UserForm_Activate()
        Dim elemento As String
        If ListBox1.ListCount >= 1 Then ListBox1.Clear
        i = 1
        Do Until Sheets("Archivio").Cells(1, i).Value = Empty
                With Sheets("Archivio")
                        elemento = .Cells(1, i).Value
                End With
                ListBox1.AddItem elemento
                i = i + 8
        Loop
        ListBox1.ListIndex = 0
        svuota_box
End Sub
```

Per indentare il codice è possibile usare un Add-In che potete trovare **a questo link**.

NB Non è ancora stata rilasciata una versione che supporti Office 64bit, per cui la versione 2013 non viene supportata, ma è possibile usare un indentatore on-line **a questo Link**

Debug utilizzando una finestra di messaggio

Uno dei metodi più elementari e spesso utilizzati durante la scrittura di codice è quello di usare una finestra di messaggio per controllare i valori mutevoli di una variabile (MsgBox). Generalmente si utilizza una finestra di messaggio immediatamente dopo la riga di codice in cui la variabile assume un valore, per verificare come la procedura viene eseguita con i valori delle variabili che cambiano dinamicamente. Per visualizzare la finestra di messaggio per ottenere il valore di una variabile si utilizza la sintassi:

MsgBox NomeVariabile

Questa riga di codice verrà rimossa dopo aver terminato la verifica del valore. Di seguito vediamo alcuni esempi che mostrano come utilizzare la funzione MsgBox per verificare i valori delle variabili che cambiano mentre viene eseguito il codice.

Esempio: Verifica dei valori delle variabili alla fine del Ciclo

```
Sub Prova1()
        Dim I, tot As Integer
        i = 0
        Do
                i = i + 1
                MsgBox i
'Restituisce 1, 2, 3
                tot = tot + i
        Loop Until i > 2
'restituisce 6
        MsgBox tot
End Sub
```

Esempio: Verifica dei valori delle variabili all'inizio del Ciclo

```
Sub Prova1()
        Dim i , tot As Integer
        For i = 1 To 2
                Do Until tot > 3
                        tot = tot + 2
                        MsgBox " i = " & i
'Restituisce i=1
                        MsgBox "tot =" & tot
'Restituisce tot=2
                Loop
                Next i
                MsgBox " Totale Complessivo = " & tot
'restituisce 4
        End Sub
```

Utilizzare i punti di interruzione

È possibile inserire uno o più punti di interruzione su qualsiasi riga del codice, che permette di fermare temporaneamente l'esecuzione della macro in quel punto. A questo punto la macro è in modalità interruzione e consente di vedere il valore corrente delle variabili spostando il cursore del mouse su di loro. Un punto di interruzione può essere posizionato su qualsiasi riga del codice, ma non sulle righe che definiscono le variabili o nella sezione generale delle dichiarazioni. I punti di interruzione sono generalmente fissati a una riga di codice specifico in cui si immagina un errore e che vengono eliminati quando sono stati risolti gli errori.

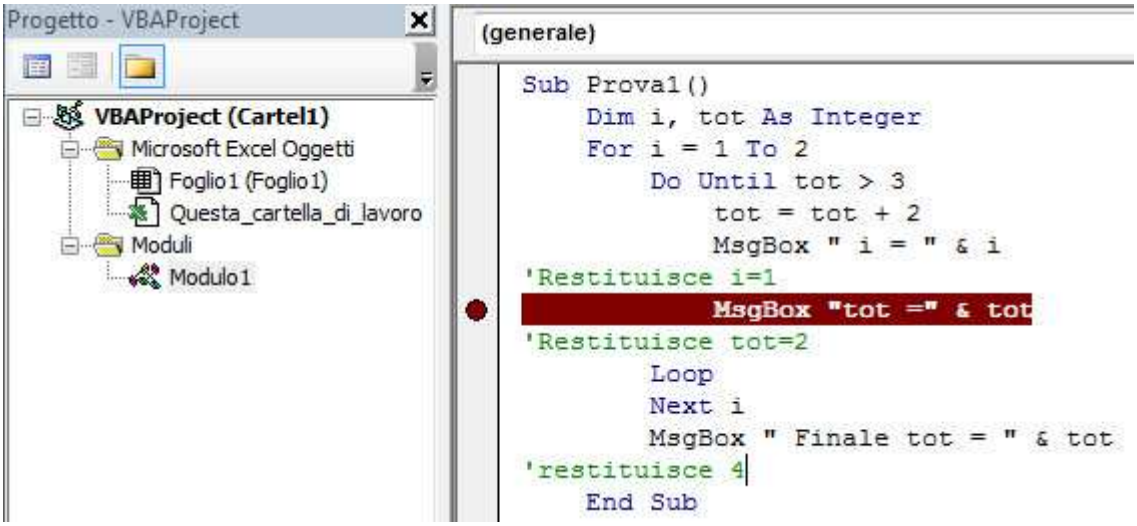


Fig. 2

Aggiungere o Cancellare un punto di interruzione

Per aggiungere un punto di interruzione, si deve operare in questo modo:

- Cliccare sul bordo sinistro della riga di codice in cui si desidera inserirlo
- Cliccare sulla riga e premere F9
- Dal menu Debug – Imposta/rimuovi punto di interruzione

Per cancellare un punto di interruzione, si deve ripetere l'operazione appena descritta, mentre se sono presenti vari punti di interruzione per rimuoverli tutti si deve premere **Ctrl + Maiusc + F9** oppure dal menu **Debug - Rimuovi punti di interruzione**.

Utilizzo della modalità Interruzione o Pausa

La macro va in modalità interruzione quando si interrompe l'esecuzione del codice e si mette in pausa temporaneamente, oppure quando

- Si incontra un punto di interruzione
- Premendo i tasti **Ctrl + Pausa** durante l'esecuzione del codice
- Se si incontra una istruzione Stop nel codice
- Al verificarsi di un errore di sintassi o di un errore Run-Time.

Interrompendo questa modalità è possibile visualizzare lo stato corrente della macro e controllare il valore delle variabili portando il cursore del mouse sulla variabile stessa. Una volta che il codice è in modalità interruzione, è possibile scegliere di continuare con l'esecuzione premendo il tasto **F5**, oppure cliccando su **Ripristina** dal menu **Esegui**. Al verificarsi di un errore, è possibile correggere l'errore e scegliere di continuare, oppure terminare l'esecuzione del codice e riavviare la macro.

Esecuzione Passo - Passo del codice

Per attivarla si deve fare clic su **Esegui istruzione** dal menu **Debug** o premere **F8**. In fase di progettazione, all'interno di una procedura, l'esecuzione del codice parte dall'inizio della macro ed entra in modalità pausa prima di eseguire la prima riga di codice. Questo è un modo per entrare in modalità pausa passando attraverso il codice che verrà eseguito una riga alla volta per poi passare alla riga successiva.

Finestra Immediata

Si può visualizzare la Finestra Immediata dal menu **Visualizza** oppure premere i tasti **Ctrl + G**. La finestra Immediata è una zona di debug primaria, ed è utilizzato per:

- Visualizzare i risultati delle istruzioni della macro
- Digitare un'istruzione o una riga di codice direttamente nella finestra e premere Invio per eseguirlo
- Modificare il valore di una variabile durante l'esecuzione di una macro

Quando la macro è in modalità di pausa, è possibile assegnare un nuovo valore alla variabile nella finestra Immediata come si farebbe nella macro stessa, inoltre nella finestra Immediata è possibile valutare le dichiarazioni VBA o le espressioni che possono essere correlate o meno alla macro. Quando la macro è in modalità di interruzione, una dichiarazione VBA nella finestra immediata viene eseguita nel contesto di tale macro, per esempio se si digita MsgBox i, dove i è il nome di una variabile utilizzata nella macro, nella finestra Immediata, si otterrà il valore corrente della variabile i come se il comando fosse stato utilizzato all'interno della macro in esecuzione.

Per restituire un valore, si deve precedere l'espressione con un punto interrogativo, per esempio, se digitiamo un'espressione come ? 100/2, viene riportato il valore della divisione dei due numeri, come è possibile vedere nell'immagine sotto riportata

Fig. 3

Per visualizzare il valore di una variabile nella finestra Immediata è possibile usare il comando **Debug.Print** che permette di vedere lo stato delle variabili e le decisioni che il nostro programma prende, in quanto con l'aggiunta di questa dichiarazione vengono stampate informazioni utili a correggere i nostri programmi nella finestra immediata. La sintassi è la seguente:

Debug.Print Expression

Ad esempio, se nel foglio di Excel nella cella A1 abbiamo il testo "Prova debug.print" e ci posizioniamo su quella cella possiamo conoscerne il contenuto scrivendo nella finestra immediata il comando ?activecell.Value che riporterà:

```
Immediata
?100/2
 50

?activecell.Value
Prova debug.print
```

Fig. 4

Questo comando può essere usato in abbinamento alla punteggiatura in questo modo: *?activecell.Value,* se inseriamo una virgola alla fine del comando, alla prossima istruzione Debug.Print il risultato viene posto sulla stessa riga usando una tabulazione predefinita, mentre se inseriamo *?activecell.Value;* con un punto e virgola alla fine al prossimo comando il risultato viene inserito sulla stessa riga attaccato a quello precedente

Finestra Locale

Possiamo utilizzare la finestra Locale per vedere tutte le variabili dichiarate nella routine corrente e il loro tipo e valore corrente. La finestra Locale visualizzerà per le variabili locali dichiarate nella routine corrente e le variabili dichiarate nella sezione di dichiarazione moduli - il diplay è in 3 colonne, *Espressione, Valore e Tipo*. Si noti che la finestra Locale viene usata solo quando la macro è in modalità Pausa.

```
Sub prova()
    Dim i, k As Integer
    For i = 1 To 5
        k = "Controllo n° " & i
        Next i
    End Sub
```

Variabili locali

VBAProject.Modulo1.prova

Espressione	Valore	Tipo
Modulo1		Modulo1/Modulo1
i	1	Variant/Integer
k	0	Integer

Fig. 5

Il valore di una variabile locale può essere modificata nella finestra Locale facendo clic sulla colonna Valore

Finestra Espressioni di controllo

Tramite questa finestra è possibile monitorare i valori delle variabili in modalità di interruzione, mentre la finestra Locale automaticamente tiene traccia di tutte le variabili dichiarate nella routine corrente, nella finestra di controllo è necessario specificare le variabili che si desidera tenere traccia. Nella finestra di controllo, indipendentemente da quella attuale, è possibile monitorare i valori delle variabili attraverso moduli e procedure. È possibile aggiungere manualmente le variabili e persino le espressioni alla finestra di controllo, quando è in modalità di interruzione.

Per aggiungere una variabile alla finestra di controllo, si deve cliccare col destro del mouse sulla variabile e selezionare Aggiungi Espressione di Controllo dal menu del tasto destro del mouse, oppure posizionare il cursore sulla variabile e selezionare Aggiungi Espressione di Controllo dal menu Debug, e in entrambi i casi, verrà visualizzata una finestra di dialogo in cui compare il campo Espressione e verrà visualizzato il nome della variabile.

Fig. 6

Per modificare o eliminare il controllo, si deve selezionare la variabile nella finestra di controllo, fare clic col destro del mouse e selezionare Elimina Espressione di controllo

Il Metodo Find in VBA

Per cercare un articolo specifico o un valore in un intervallo, si può utilizzare il metodo *Find* che restituisce il Range, vale a dire, la cella, dove si trova l'elemento o valore. Se non viene trovata nessuna corrispondenza viene restituito *Nothing*.

Sintassi: *RangeObject.Find(What, After, LookIn, LookAt, SearchOrder, SearchDirection, MatchCase, MatchByte, SearchFormat)*

E' necessario specificare solo l'argomento *What*, tutti gli altri sono facoltativi e rappresentano:

RangeObject: Rappresenta un intervallo in cui viene cercato l'elemento o lo specifico valore ed è possibile cercare all'interno di celle specifiche, vale a dire, Range, Colonne o le celle di un foglio di lavoro.

What: E 'la voce o il valore che viene ricercato e può essere di qualsiasi tipo di dati.

After: Rappresenta una singola cella che è necessario specificare, dopo di che la ricerca inizia. Perché la ricerca inizia dopo questa cella, la cella specificata viene cercata alla fine e quando la ricerca inizia dopo la cella specificata e raggiunge la fine dell'intervallo di ricerca, senza trovare il valore di ricerca, la ricerca ricomincia dall'inizio dell'intervallo di ricerca fino alla cella specificata. Se l'argomento non viene specificato, la cella iniziale è definita nell'angolo superiore sinistro del campo di ricerca, dopo di che inizia la ricerca.

Se si specifica *After: = Range ("A13")* in cui il campo di ricerca è *Range ("A1: A20")*, il metodo *Find* inizierà la ricerca dalla cella A14, fino alla cella A20 e successivamente cercherà dalla cella A1 fino alla cella A13.

Lookin: Questo argomento specifica il tipo di informazioni - può essere *xlValues* o *xlFormulas* o *xlComments* che indicano il tipo di valore da cercare, se una formula, un commento o un valore. Il valore predefinito è *xlFormulas*.

	A
1	12
2	24
3	87
4	78
5	98
6	82
7	176
8	57
9	176
10	90
11	76
12	323
13	432
14	45
15	234
16	somma
17	77
18	99
19	9
20	25

Fig. 1

Prendiamo il caso in cui la cella A7 di *figura 1* contenga la formula *=SOMMA(A4;A5)* e il totale della somma è 176, si può usare il metodo Find per cercare il valore nelle formule in questo modo:

ActiveSheet.Range("A1:A20").Find(What:="176", After:=ActiveSheet.Range("A1"), LookIn:=xlFormulas)

Verrà restituita la cella *$A$9*, perché il valore 176 NON compare nella formula della cella A7, ma compare come valore nella cella A9. Se invece usiamo il metodo Find per ricercare il valore 176 deve essere espresso in questo modo:

ActiveSheet.Range("A1:A20").Find(What:="176", After:=ActiveSheet.Range("A1"), LookIn:=xlValues)

Restituirà *$A$7*, in quanto il valore 176 appare come risultato della formula nella cella A7 e solo dopo nella cella A9. Analogamente, nel caso in cui la cella A7 contiene la formula *=SOMMA(A4;A5)*, e la cella A16 contiene la stringa *somma*, si può applicare il metodo Find in questo modo:

ActiveSheet.Range("A1:A20").Find(What:="somma", After:=ActiveSheet.Range("A1"), LookIn:=xlValues)

Restituirà *$A$16*, mentre invece se il metodo viene espresso nel seguente modo:

ActiveSheet.Range("A1:A20").Find(What:="sum", After:=ActiveSheet.Range("A1"), LookIn:=xlFormulas)

Restituirà *$A$7*

Esempio: Le opzioni del metodo Find

```
Sub cerca_1()
Dim valoreC As Range
'Restituisce $A$9
Set valoreC = ActiveSheet.Range("A1:A20").Find(What:="176",
After:=ActiveSheet.Range("A1"), LookIn:=xlFormulas)
If Not valoreC Is Nothing Then
MsgBox valoreC.Address
'End If
'Restituisce $A$7
Set valoreC = ActiveSheet.Range("A1:A20").Find(What:="176",
After:=ActiveSheet.Range("A1"), LookIn:=xlValues)
If Not valoreC Is Nothing Then
MsgBox valoreC.Address
'End If
'Restituisce $A$16
Set valoreC = ActiveSheet.Range("A1:A20").Find(What:="somma",
After:=ActiveSheet.Range("A1"), LookIn:=xlValues)
If Not valoreC Is Nothing Then
MsgBox valoreC.Address
End If
'Restituisce $A$7
Set valoreC = ActiveSheet.Range("A1:A20").Find(What:="sum",
After:=ActiveSheet.Range("A1"), LookIn:=xlFormulas)
If Not valoreC Is Nothing Then
MsgBox valoreC.Address
End If
End Sub
```

E' possibile utilizzare anche gli altri parametri della sintassi del metodo Find come:

LookAt: È possibile specificare *xlWhole* o *xlPart* , se volete, rispettivamente, una corrispondenza esatta o una corrispondenza parziale. Una ricerca utilizzando xlPart per "Gianni" restituirà la cella che ha "Gianni Morandi", perché c'è una corrispondenza parziale. Utilizzare xlWhole per abbinare l'intero valore o la stringa che corrisponde esattamente al valore di una cella. Il valore predefinito è xlPart.

SearchOrder: È possibile specificare *xlByRows* o *xlByColumns* per questo argomento, che indicano se cercare per righe o per colonne. Il valore predefinito è xlByRows.

Supponiamo che le celle A7 e B3 contengano la stringa "somma" si può utilizzare il metodo Find per una ricerca nelle righe in questo modo:

ActiveSheet.Range("A1:B20").Find (What:= "somma", LookIn:=xlValues, SearchOrder:=xlByRows)

Mentre invece per una ricerca nelle colonne si deve applicare il metodo Find in questo modo:

ActiveSheet.Range ("A1:B20").Find (What:= "somma", LookIn: = xlValues, SearchOrder: = xlByColumns)

XlSearchDirection: È possibile specificare *xlNext* per ricerche verso il basso (cioè il valore corrispondente successivo) o *xlPrevious* per ricerche verso l'alto o all'indietro (cioè il valore corrispondente precedente) nel campo di ricerca. Il valore predefinito è xlNext. Se si specifica After: = Range ("A13") in cui il campo di ricerca è Range ("A1: A20") e impostare il SearchDirection: = xlNext , allora la funzione di ricerca inizierà a cercare dalla cella A14 fino alla cella A20 e poi ricercare dalla cella A1 fino alla cella A13.

MatchCase : Si deve specificare il valore *True* per una ricerca case-sensitive. Il valore predefinito è False.

MatchByte: Questo argomento può essere utilizzata solo se si seleziona il supporto delle lingue a doppio byte.

SearchFormat: Indica se si desidera cercare una specifica formattazione con il valore True o False. Il valore predefinito è False. Si deve specificare il formato utilizzando la proprietà *FindFormat* dell'oggetto Application, e impostare l'argomento *SearchFormat true*. Vedi l'esempio sotto riportato che illustra questa tesi.

Esempio: Utilizzare il metodo Find per cercare la prima occorrenza del valore stringa "somma", che è in grassetto. Si noti che la ricerca inizierà dopo la cella A1 (cioè dalla A2) in assenza dell'argomento After

```
Sub cerca_formato()
Dim cerca_1 As Range, ultima As Range, cerca_val As Range
'impostare l'intervallo di ricerca
Set cerca_1 = ActiveSheet.Range("A1:A20")
'specificare l'ultima cella del range
Set ultima = cerca_1.Cells(cerca_1.Cells.Count)
'Specificare il formato utilizzando FindFormat
Set cerca_val = cerca_1.Find(What:="somma", After:=ultima, LookIn:=xlValues,
SearchFormat:=True)
MsgBox cerca_val.Address
End Sub
```

Ogni volta che il metodo Find viene utilizzato, le impostazioni per LookIn, LookAt, SearchOrder, e MatchByte vengono salvate, a meno che i valori per questi argomenti non siano specificati di nuovo, i valori precedentemente salvati vengono utilizzati di nuovo la prossima volta che viene richiamato il metodo. Quindi è importante impostare questi argomenti **in modo esplicito** ogni volta che questo metodo viene utilizzato. In questo modo, il valore di argomento utilizzato in precedenza diventa il default se non specificato nel successivo uso.
Ad esempio, se si specifica LookIn con l'argomento xlFormulas, poi xlFormulas diventa il valore predefinito per l'argomento LookIn. E nel successivo utilizzo, se si omette di menzionare l'argomento LookIn, si imposterà xlFormulas.

Nel caso in cui si desidera cercare un elemento o un valore in un intervallo, una pratica comune è utilizzare un ciclo come Loop o For Next. Se l'intervallo di ricerca è piuttosto grande, l'uso del ciclo potrebbe richiedere molto tempo, mentre invece un grande vantaggio nell'uso del metodo Find è la sua velocità.
Per trovare occorrenze multiple di un elemento o un intervallo si può utilizzare il metodo *FindNext* o *FindPrevious*. Questi metodi sono utilizzati per proseguire la ricerca con il metodo Find, utilizzando gli stessi parametri o condizioni, e restituire il successivo (metodo FindNext) o precedente (metodo FindPrevious) cella corrispondente.

- **Il Metodo Range.FindNext**: Sintassi: *RangeObject.FindNext (After)*

RangeObject: Rappresenta un intervallo in cui viene cercato l'elemento o il valore specifico.

After: Rappresenta una singola cella che si deve specificare dopo la quale inizia la ricerca. Quando la ricerca inizia dopo la cella specificata e raggiunge la fine dell'intervallo di ricerca, senza trovare il valore di ricerca, la ricerca ricomincia dall'inizio dell'intervallo di ricerca fino alla cella specificata. È facoltativo specificare questo argomento, e se non specificato, è la cella nell'angolo superiore sinistro del campo di ricerca, dopo di che inizia la ricerca.

- **Il Metodo Range.FindPrevious**: Sintassi: *RangeObject.FindPrevious (After)*

RangeObject: Rappresenta un intervallo in cui viene cercato l'elemento o valore specifico.

After: Rappresenta una singola cella che si specifica **prima** che inizia la ricerca, perché la ricerca inizia da questa cella. È facoltativo specificare questo argomento, e se non specificato, è la cella nell'angolo superiore sinistro del campo di ricerca prima che inizia la ricerca.

Esempio: Trovare più occorrenze di un valore in un intervallo; trovare la stringa "vecchia" in un campo di ricerca, sostituirla con "nuova" e cambiare il colore del carattere.

```
Sub multi_cerca()
Dim cerca1 As Range, ultima As Range, cerca_prima As Range
Dim primo_ind As String
'Impostare l'intervallo di ricerca
Set cerca1 = ActiveSheet.Range("A1:A100")
'Specifica ultima cella nel range
Set ultima = cerca1.Cells(cerca1.Cells.Count)
'Trovare la stringa "vecchia" nel campo di ricerca
Set cerca_prima = cerca1.Find(What:="vecchia", After:=ultima, LookIn:=xlValues,
LookAt:=xlWhole, SearchOrder:=xlByRows, SearchDirection:=xlNext, MatchCase:=False,
SearchFormat:=False)

'Se "vecchia" si trova nel campo di ricerca
If Not cerca_prima Is Nothing Then
'Salva l'indirizzo del primo risultato di "vecchia", nella variabile primo_ind
primo_ind = cerca_prima.Address
Do
'Trova la successiva occorrenza di "vecchia". Si noti, che non si parte dalla prima
occorrenza di "vecchia"
Set cerca_prima = cerca1.FindNext(cerca_prima)
'Sostituire "vecchia" con "nuova"
cerca_prima.Value = "nuova"
'Colore del carattere è cambiato
cerca_prima.Font.Color = vbRed

Loop Until cerca_prima.Address = primo_ind
End If
End Sub
```

Esempio: Con riferimento all'Immagine 2, sotto riportata, il codice proposto mostra come utilizzare il metodo Find e la proprietà Offset per fare una ricerca verticale.

	A	B	C	D	E	F	G
1	Inserisci Voti				Tabella Studenti		
2	Studente	Voto			Studente	Classe	Voto
3	Luca				Luca	4	55
4	Carlo	74			Carlo	3	74
5	Ivan				Ivan	3	68
6	Marco	82			Marco	5	82
7	Gianni				Gianni	4	78
8							

Fig. 2

Per ogni studente nella colonna A, si deve trovare il nome dello stesso nella colonna E, e inserire i suoi voti nella colonna B, solo se è nella classe 3

```
Sub cerca_2()
Dim cerca1 As Range, nomeSt As Range, prima As Range, stud As Range
Set cerca1 = ActiveSheet.Range("E3:E7")
Set nomeSt = ActiveSheet.Range("A3:A7")
'si ricercano tutti i nomi degli studenti menzionati nell'intervallo specificato
For Each stud In nomeSt
Set prima = cerca1.Find(What:=stud, LookIn:=xlValues, LookAt:=xlWhole,
SearchOrder:=xlByRows, SearchDirection:=xlNext, MatchCase:=False,
SearchFormat:=False)
'Se il nome dello studente viene trovato e se studente è in classe 3
If Not prima Is Nothing And prima.Offset(0, 1) = "3" Then
stud.Offset(0, 1) = prima.Offset(0, 2)
End If
Next
End Sub
```

Utilizzo del metodo Find per cercare una data

Excel memorizza tutte le date come numeri interi e il tempo come frazioni decimali. Con questo sistema, Excel può aggiungere, sottrarre o confrontare date e ore, proprio come qualsiasi altro numero, considerando come data iniziale di questo sistema il 1/1/1900 0:00:00, che Excel considera erroneamente il 1900 come un anno bisestile, si presume che questo errore sia stato fatto consapevolmente da Microsoft per garantire la compatibilità con Lotus 1-2-3, e quindi in realtà il bug sarebbe stato in Lotus 123 (predecessore di Excel).

In Excel, la data equivale a un "numero di serie" (che è un valore numerico), che è il conteggio del numero di giorni trascorsi da una certa data di riferimento. La parte intera (valori a sinistra del separatore decimale) è il numero di giorni trascorsi dal 1 gennaio 1900, ad esempio, 1 gennaio 1900 viene memorizzato come 1, il 2 gennaio 1900 viene memorizzato come 2, il 15 marzo 2001 viene memorizzato come 36.965. La parte frazionaria (valori a destra del decimale) contiene informazioni temporali, e rappresenta il tempo come frazione di un giorno intero. Ad esempio, 12:00 (mezzanotte) è memorizzato come 0, 06:00 viene memorizzato come 0,25, 12:00 (mezzogiorno) viene memorizzato come 0.5, 06:00 viene memorizzato come 0,75, 06:00:30 viene memorizzato come 0,750347222. Per controllare il "numero di serie" di una data e ora si deve formattare la cella come "Generale".

L'utilizzo del metodo Find per trovare o cercare una data può essere difficile, il formato della data deve corrispondere al formato data predefinito, come impostato nel vostro desktop, che, se non specificatamente modificato, dovrebbe essere nel suo formato standard di "data breve" o "data lunga", vale a dire, " 22/01/2010" o "22 gennaio 2010". Non importa in quale data il formato viene visualizzato nel foglio di lavoro, deve solo essere una data valida per Excel corrispondente ad un numero di serie valido. Il codice seguente mostra come utilizzare il metodo Find per cercare una data

Esempio: Ricerca di una data all'interno di un intervallo

	A	B	C	D
1	2-mar-10			36254
2	10-feb-75			
3	16-set-11			
4	22-ago-10			
5	4-apr-99			
6	2-feb-85			
7				

Fig. 3

In questo esempio l'utente inserisce la data che vuole trovare nella cella D2 che viene rappresentata col valore di 36254 in quanto la cella è formattata come "Generale". La ricerca avviene nella colonna A.

```
Sub cerca_data()
Dim primo As Range, cerca1 As Range, ultimo As Range
Dim strDate As String
'intervallo di ricerca
Set cerca1 = ActiveSheet.Range("A1:A20")
Set ultimo = cerca1.Cells(cerca1.Cells.Count)

'si stabilisce che la data da cercare è in D2
strDate = Format(ActiveSheet.Range("D1"), "Short Date")
'Format ("4/4/99", "Short Date") restituisce "04/04/1999"
'Format ("4/4/99", "Long Date") restituisce "Domenica 4 Aprile 1999".
'La funzione IsDate restituisce True se l'espressione è una data valida, altrimenti restituisce False.
If IsDate(strDate) = False Then
MsgBox "Formato Data Incorretto"
Exit Sub
End If
'CDate converte un numero o una stringa di testo in un tipo di dati Date.
'CDate (36240) restituisce "04/04/1999"

Set primo = cerca1.Find(What:=CDate(strDate), After:=ultimo, LookIn:=xlFormulas,
LookAt:=xlWhole, SearchOrder:=xlByRows, SearchDirection:=xlNext, MatchCase:=False,
SearchFormat:=False)
If Not primo Is Nothing Then
'Se la data viene trovata si stampa l'indirizzo della cella (A5 in questo esempio)
MsgBox primo.Address
Else
MsgBox "Data non trovata"
End If
End Sub
```

Il Metodo OnTime ed esempi sui colori

Esempi di codice per modificare il colore del carattere in celle aventi "formule", "formule e numeri" e "costanti".

```
Sub colore_formule()
Dim coloreF As Long
Dim coloreN As Long
Dim coloreC As Long
Dim cell As Range
coloreF = RGB(Red:=0, Green:=255, Blue:=0)
coloreN = RGB(Red:=0, Green:=0, Blue:=0)
coloreC = RGB(Red:=0, Green:=0, Blue:=255)
For Each cell In ActiveSheet.UsedRange.SpecialCells(xlCellTypeFormulas)
'celle con formule
cell.Font.Color = coloreF
Next cell
For Each cell In ActiveSheet.UsedRange.SpecialCells(xlCellTypeFormulas, xlNumbers)
'celle con numeri
cell.Font.Color = coloreN
Next cell
For Each cell In ActiveSheet.UsedRange.SpecialCells(xlCellTypeConstants)
'celle con costanti (non formule)
cell.Font.Color = coloreC
Next cell
End Sub
```

```
Sub colore_formule2()
Dim coloreF As Long
Dim coloreN As Long
Dim coloreC As Long
Dim cell As Range
coloreF = RGB(Red:=0, Green:=255, Blue:=0)
coloreN = RGB(Red:=0, Green:=0, Blue:=0)
coloreC = RGB(Red:=0, Green:=0, Blue:=255)
For Each cell In ActiveSheet.UsedRange
If cell.HasFormula = True Then
'celle con formule
cell.Font.Color = coloreF
If IsNumeric(cell) = True Then
'celle con numeri
cell.Font.Color = coloreN
End If
Else
cell.Font.Color = coloreC
'cellule con costanti
End If
Next cell
End Sub
```

Il codice sotto riportato cambia il colore di una cella quando vengono immessi i valori delle celle sfruttando l'evento Change del foglio di lavoro

```
Private Sub Worksheet_Change(ByVal Target As Range)
Dim coloreF As Long
Dim coloreN As Long
Dim coloreC As Long
coloreF = RGB(Red:=0, Green:=255, Blue:=0)
coloreN = RGB(Red:=0, Green:=0, Blue:=0)
coloreC = RGB(Red:=0, Green:=0, Blue:=255)
With Target
If .HasFormula Then
'celle con formule
.Font.Color = coloreF
If IsNumeric(Target) Then
'celle con numeri
.Font.Color = coloreN
End If
Else
.Font.Color = coloreC
'celle con costanti (non formule)
End If
End With
End Sub
```

Eseguire le macro a intervalli periodici o a un tempo determinato

Il Metodo Application.OnTime

Si può utilizzare il metodo *Application.OnTime* per eseguire una procedura a intervalli specifici o in un momento specifico della giornata. *Sintassi*: *ApplicationObject OnTime (earliestTime, ProcedureName, LatestTime, Schedule).*

Usando questo metodo è possibile pianificare l'esecuzione di una procedura in futuro in specifici intervalli di tempo, a partire da oggi, oppure è possibile fissare un momento specifico della giornata. L' oggetto Application rappresenta l'intera applicazione Excel, ed è l'oggetto più in alto nella gerarchia degli oggetti di Excel. Gli argomenti *earliestTime* e *ProcedureName* devono essere specificati, mentre gli altri argomenti sono facoltativi.

earliestTime: Questo argomento specifica il momento in cui la procedura viene eseguita

ProcedureName: Questo argomento specifica il nome della procedura che si desidera eseguire.

LatestTime: Con questo argomento è possibile impostare il limite di tempo per l'esecuzione della procedura vale a dire che se si imposta *LatestTime* a*earliestTime + 20* e se nel frattempo un'altra procedura viene eseguita ed Excel non è "pronto" entro 20 secondi, questa procedura non verrà eseguita, tralasciando l'argomento LatestTime Excel eseguirà la procedura. Tralasciando l'argomento *Schedule* sarà impostato di default su True, che stabilisce una nuova procedura Ontime. Per annullare una procedura OnTime esistente impostata in precedenza, si deve specificare OnTime su False.

Per eseguire una procedura in specifici intervalli a partire da oggi, si deve utilizzare "*Now + TimeValue (ora)*", se per esempio si esprime il metodo con una espressione come: *TimeValue ("20:30:00")*, verrà

eseguita una procedura alle 20:30, mentre invece per eseguire una procedura a intervalli specifici di tempo (ad esempio, da oggi), si deve utilizzare *Now + TimeValue (ora)* vale a dire, *Ora + TimeValue ("00:00:05")* imposta l'intervallo di tempo a 5 secondi, al quale intervallo la procedura verrà eseguita.

Arrestare o annullare una procedura utilizzando il metodo OnTime

Se si tenta di chiudere la cartella di lavoro, mentre una procedura è in esecuzione utilizzando Application.OnTime, Excel riaprirà la cartella di lavoro per completare la procedura, quindi, sarà necessario annullare la procedura. Per annullare una procedura in esecuzione utilizzando il metodo OnTime, è richiesto il tempo preciso della sua esecuzione pianificata. Si noti che se non si passa il tempo a una variabile, Excel non sa quale metodo OnTime deve utilizzare per annullare l'operazione. Per esempio se viene usata la sintassi *Now + TimeValue ("00:00:03")*, non è un valore statico, ma lo può diventare quando viene passato a una variabile. Ciò significa che il momento in cui la procedura è da eseguire (argomento earliestTime) deve essere assegnato a una variabile, utilizzando una variabile pubblica per rendere la variabile a disposizione di tutte le procedure in tutti i moduli e poi utilizzarlo per annullare il metodo OnTime.

Esempio: Questa procedura utilizza il metodo OnTime il cui valore di incremento automatico viene reperito in una cella e viene eseguita a intervalli di tempo specifici. Il procedimento si Arresta al superamento di un valore in una cella specifica. La procedura deve essere inserita in un modulo standard

```
Public tempoR As Date
Sub incremento1()
'imposta l'intervallo di tempo a 3 secondi
tempoR = Now + TimeValue("00:00:03")
'la procedura verrà eseguita in automatico nell'intervallo di tempo fissato
Application.OnTime EarliestTime:=tempoR, Procedure:="incremento1", schedule:=True
'incrementare il valore nella cella A1 di 5 ogni volta che si esegue la macro
Cells(1, 1).Value = Cells(1, 1).Value + 5
'la procedura si ferma quando trova il valore 25 nella cella A1
If Cells(1, 1).Value > 25 Then
'annullare la procedura impostando l'argomento schedule a False
Application.OnTime tempoR, "incremento1", , False
End If
End Sub
```

Esempio: Questa procedura utilizza il metodo OnTime il cui valore di incremento automatico viene reperito in una cella e viene eseguita a intervalli di tempo specifici. Il procedimento si Arresta dopo essere stata eseguita un determinato numero di volte. La procedura deve essere inserita in un modulo standard

```
Public tempoI As Date
Dim conta As Integer
Sub incremento2()
tempoI = Now + TimeValue("00:00:03")
Application.OnTime tempoI, "incremento2", , True
Cells(1, 1).Value = Cells(1, 1).Value + 5
conta = conta + 1
'Interrompere la procedura dopo averla eseguita per 5 volte
If conta = 5 Then
Application.OnTime tempoI, "incremento2", , False
conta = 0
End If
End Sub
```

Esempio: Avviare la procedura OnTime automaticamente quando la cartella di lavoro viene aperta e interromperla automaticamente alla chiusura della cartella di lavoro. Questa procedura imposta i promemoria in orari specifici e chiude automaticamente la cartella di lavoro in un momento specifico. Aggiungere il codice sotto riportato al modulo ThisWorkbook

```
Private Sub Workbook_Open()
ricordami
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
On Error Resume Next
'fermo il promemoria
ricordamiST
'salva cartella di lavoro prima della chiusura
ThisWorkbook.Save
End Sub
```

Le procedure sotto riportate devono essere inserite in un modulo standard:

```
Public dTime As Date
Sub ricordami()
On Error Resume Next
dTime = Now + TimeValue("00:00:01")
Application.OnTime dTime, "ricordami"
'Chiudere la cartella di lavoro nel momento specificato

If Time = TimeSerial(18, 30, 0) Then
chiudiCAR
End If

'Impostare promemoria di chiusura ufficio
If Time = TimeSerial(17, 30, 0) Then
Application.OnTime dTime, "chiudiOF"
End If

'impostare promemoria pausa pranzo
If Time = TimeSerial(13, 0, 0) Then
Application.OnTime dTime, "pranzo"
End If

'impostare il promemoria per la pausa caffè
If Time = TimeSerial(11, 15, 0) Then
Application.OnTime dTime, "Pcaffe"
End If
End Sub
```

```
Sub ricordamiST()
'fermare la procedura ricordami
Application.OnTime dTime, "ricordami", , False
End Sub

Sub chiudiCAR()
On Error Resume Next
ricordamiST
'salva cartella di lavoro prima della chiusura
ThisWorkbook.Save
'chiude la cartella di lavoro
ThisWorkbook.Close
End Sub

Sub Pcaffe()
Dim obj As Object
Dim strMsg As String
Set obj = CreateObject("WScript.Shell")
'riproduco un segnale acustico di avviso
Beep
'Popup Message Box si chiuderà automaticamente da solo
strMsg = obj.Popup("Pausa Caffè!", vbOKCancel)
End Sub

Sub pranzo()
Dim obj As Object
Dim strMsg As String
Set obj = CreateObject("WScript.Shell")
Beep
strMsg = obj.Popup("Pausa Pranzo!", vbOKCancel)
End Sub

Sub chiudiOF()
Dim obj As Object
Dim strMsg As String
Set obj = CreateObject("WScript.Shell")
Beep
strMsg = obj.Popup("Chiudi Ufficio!", vbOKCancel)
End Sub
```

Importare dati da un file di testo in Excel con VBA

Essere in grado di manipolare i file di testo e/o CSV con VBA è una competenza che in programmazione risulta molto utile e in questa sezione verrà illustrato il modo per poterlo fare. Un file CSV è composto da dati separati tra di loro da virgole, infatti l'acronimo **CSV** indica Comma Separated Value, ovvero valori separati da virgole e i dati terminano con un ritorno a capo premendo il tasto Invio sulla tastiera (Fig. 1), mentre se un file ha ogni voce su una riga separata con il carattere di tabulazione, allora si dice che sia un file TXT (Fig. 2)

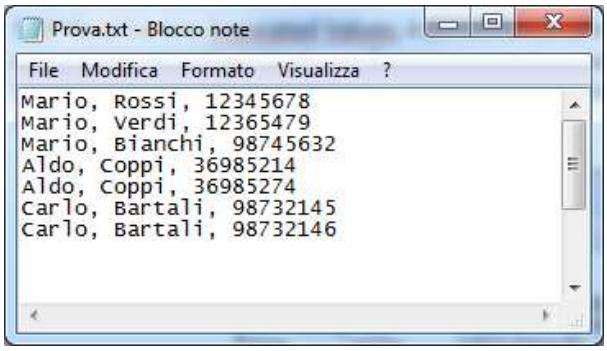


Fig. 1

Fig. 2

Il formato CSV generalmente viene utilizzato per importare e esportare i dati presenti in una tabella e ogni linea di testo del file rappresenta una riga della tabella, mentre le linee rappresentano i campi, divisi da un carattere separatore, che rappresentano i valori presenti nelle varie colonne. È possibile, naturalmente, aprire un file di testo direttamente da Excel, basta seguire il percorso **Dati - Carica dati esterni** dalla sezione Opzioni Testo della barra multifunzione di Excel e verrebbe in tal modo richiamata l'importazione guidata del testo. Tuttavia, è utile sapere che è possibile farlo a livello di programmazione usando VBA. Per meglio comprendere il procedimento in questa lezione andremo ad aprire il file CSV illustrato in *figura 1* spostando il codice numerico che si vede all'estrema destra in *figura 1* nella prima colonna.

Come prima operazione si apre Excel, si crea una nuova cartella di lavoro vuota, si clicca sulla colonna A e si formatta come testo, questo perché il codice nel file di testo è in formato numerico e se si lascia la colonna A formattata come generale (di default), si può ottenere un numero "strano". A questo punto si apre l'Editor di VBA per arrivare alla finestra del codice e si crea una nuova Sub chiamandola *ApriFile* e come prima riga di codice, si aggiunge la seguente:

```
Dim percorso As String
```

Questo codice imposta solo una variabile chiamata *percorso*, ora abbiamo bisogno di conoscere la posizione del file *Prova.csv*. Supponiamo di aver messo il file .CSV nella cartella Documenti, in questo caso è possibile utilizzare il comando **Application.DefaultFilePath** per conoscere il percorso predefinito dove vengono alloggiati i file dal sistema, a questo punto bisogna aggiungere solo il nome del file, preceduto da un backslash:

```
percorso = Application.DefaultFilePath & "\Prova.csv"
```

Se invece il file è stato inserito in qualche altra cartella allora si può fare qualcosa di simile a questo

```
percorso = "C:\Users\user\Documents\Test\Prova.csv"
```

Con questo listato il programma punta a una cartella denominata *Test* che si trova nella cartella *C: \ Users\user*, oppure si deve modificare il percorso per puntare alla cartella in cui è stato salvato il file Prova.csv.

Ora si deve aprire il file e si inizia con la parola chiave **Open**, quindi si specifica il nome del file, una modalità e un numero di file in questo modo

```
Open FileName For Mode As FileNumbe
```

La modalità di cui sopra deve essere sostituita da una delle seguenti operazioni:

- Append: viene utilizzato per aggiungere dati ad un file esistente
- Output: viene usato per scrivere in un file
- Input: viene utilizzato per leggere un file
- Binary: viene usato per leggere o scrivere dati in formato binario
- Random: viene utilizzato per inserire caratteri in un buffer di dimensioni set

Le modalità che ci interessano sono *Input* e *Output* e l'argomento *FileNumber* può essere qualsiasi numero compreso tra 1 e 511 e si precede il numero con il carattere **#**, quindi, se si sta aprendo un file a cui è stato assegnato un indice **#1**, se si apre un secondo file diverso il suo FileNumber sarebbe **# 2**, e così via, quindi il codice diventa:

```
Open percorso For Input As #1
```

Il file che vogliamo aprire è quello che abbiamo memorizzato nella variabile chiamata percorso, si tenga presente che è possibile digitare l'intero percorso del file in questo enunciato, racchiudendolo tra doppi apici:

```
Open "C:\Users\user\Documents\Test\Prova.csv" For Input As #1
```

Con questa istruzione abbiamo accesso al file in sola lettura, la riga successiva è quella di impostare una variabile per i numeri di riga, nel nostro caso possiamo usare una variabile come questa.

```
Nriga = 0
```

Al momento, abbiamo solo detto al VBA di aprire il file, non abbiamo eseguito nessuna azione sul file. Di solito come prima azione si dovrebbe eseguire la lettura dei dati nel file, useremo un ciclo *Do Until* per questo:

```
Do Until EOF (1)
.........
Loop
```

Si noti la condizione del Loop: **EOF** (1), che significa **End Of File**, mentre 1 tra parentesi tonde è il numero di file specificato in precedenza. All'interno del ciclo, inseriamo una linea di codice come la seguente:

```
Line Input # 1, LineaFile
```

Le prime tre voci prima della virgola si riferiscono alla lettura di una singola linea dal numero di file (# 1) e dopo la virgola, si indica a VBA dove si desidera posizionare questa linea, cosa che abbiamo deciso di fare tramite la variabile *LineaFile*. Ad ogni iterazione del Loop, verrà letta una nuova linea dal file di testo e posta nella variabile, tuttavia la linea avrà ancora tutte le virgole, quindi la prima linea sarà:

Mario, Rossi, 12345678

A questo punto è necessario analizzare le righe del file di testo in qualche modo, sia per togliere le virgole che per inserire i valori nelle righe del file Excel.

Un buon modo per analizzare una linea è usando la funzione **Split** tramite la quale è possibile inserire ogni elemento di una riga in un array, cosa che possiamo fare con un codice come il seguente:

```
RigaF = Split (LineaFile, ",")
```

Se osserviamo la riga di codice sopra esposto, notiamo che tra le parentesi tonde della funzione *Split*, abbiamo inserito la variabile *LineaFile* (che contiene la linea che vogliamo dividere), subito dopo è presente una virgola, e infine abbiamo il separatore che vogliamo cercare, nel nostro caso il separatore è la virgola. Quando è terminato il Loop, tramite la funzione Split otterremmo un array chiamato *RigaF* che conterrà tutte le linee del file pulite dal carattere separatore. Si deve tenere presente che il file di testo ha sempre tre elementi per riga (nome, cognome, codice), così sappiamo che l'array va da 0 a 2 posizioni. Ora possiamo andare avanti e mettere ogni elemento in una cella del foglio di calcolo in questo modo:

```
ActiveCell.Offset (Nriga, 0) .Value = RigaF (2)
ActiveCell.Offset (Nriga, 1) .Value = RigaF (1)
ActiveCell.Offset (Nriga, 2) .Value = RigaF (0)
```

Tra le parentesi tonde della funzione *Offset* abbiamo il numero di riga e il numero di colonna e stiamo usando la variabile chiamata Nriga per identificare le righe che devono ricevere i dati. Questa variabile è stata impostata a 0 in precedenza (che incrementeremo a breve), mentre invece le colonne sono sempre compensate a 0, 1 e 2, per cui un valore pari a 0, lo ricordiamo, si mantiene nella stessa colonna, un valore pari a 1 si sposta di 1 colonna, e un valore di 2 si sposta di 2 colonne dalla cella attiva.
A destra del segno di uguale, abbiamo il nostro array *RigaF*, poiché vogliamo che il codice numerico sia posto nella colonna A, abbiamo usato RigaF (2) che indica l'ultimo valore della riga letta nel file di testo e posto come primo valore nel foglio di lavoro dopo il segno di uguale. Abbiamo poi inserito RigaF (1) , che conterrà l'ultimo nome della riga del file di testo e infine, abbiamo RigaF (0) , che conterrà il primo nome della riga del file di testo. L'ultima cosa che rimane da fare all'interno del ciclo è quella di incrementare la variabile Nriga, altrimenti saremo bloccati sulla prima riga del foglio di calcolo, usando questo codice.

```
Nriga = Nriga + 1
```

Quando si apre un file, è necessario chiuderlo da qualche parte nel codice. Questo è abbastanza semplice, lo possiamo fare in questo modo:

```
Close # 1
```

Si digita la parola Close e poi, dopo uno spazio, il numero di file che si sta cercando di chiudere, in pratica tutto il codice dovrebbe essere simile a questo:

```
Sub ApriFile()
Dim percorso As String
    percorso = "C:\Users\user\Documents\Test\Prova.csv"
        Open percorso For Input As #1
Nriga = 0
Do Until EOF(1)
Line Input #1, LineaFile
RigaF = Split(LineaFile, ",")
ActiveCell.Offset(Nriga, 0).Value = RigaF(2)
ActiveCell.Offset(Nriga, 1).Value = RigaF(1)
ActiveCell.Offset(Nriga, 2).Value = RigaF(0)
Nriga = Nriga + 1
Loop
Close #1
End Sub
```

Provate il codice assicurandovi che la cella attiva del foglio di calcolo sia la cella A1 (la colonna A è quella che abbiamo formattato come testo), cliccate poi su un punto qualsiasi all'interno della Sub e premete F5 per eseguirla. Se poi tornate al foglio di lavoro si dovrebbero vedere i dati che sono stati importati dal file di testo

	A	B	C	D
1	12345678	Rossi	Mario	
2	12365479	Verdi	Mario	
3	98745632	Bianchi	Mario	
4	36985214	Coppi	Aldo	
5	36985274	Coppi	Aldo	
6	98732145	Bartali	Carlo	
7	98732146	Bartali	Carlo	
8				

Fig. 3

Scrittura di file di testo in Excel VBA

Abbiamo appena visto come importare i dati da un file di testo con VBA, ora vediamo come scrivere dei dati da un foglio di lavoro in un file di testo, prendendo come base dati quelli elencati in *Fig. 3* e scrivere di nuovo in un file Txt. Il primo compito è quello di trovare un modo per fare riferimento alle celle che ci interessano del foglio di lavoro che come vediamo è composto da 3 colonne e 7 righe, potremmo usare 2 cicli, 1 per scorrere le righe e un altro per scorrere le colonne.

```
For I = 1 To 7
For j = 1 To 3
Next j
Next i
```

Tuttavia, supponiamo che abbiamo aggiunto altre righe e colonne al foglio di calcolo e vorremmo un automatismo che calcoli quante righe e colonne contengano i nuovi dati. Il metodo migliore è quello di ottenere l'ultima riga con i dati in essa contenuti e l'ultima colonna, allora potremmo modificare i cicli in questo modo:

```
For I = 1 To UltimaR
For j = 1 To UltimaC
Next j
Next i
```

La domanda è: come possiamo ottenere i valori delle variabili *UltimaR* e *UltimaC*? Ci sono diverse metodi per trovare l'ultima riga e l'ultima colonna con i dati, un modo popolare per ottenere l'ultima riga con i dati, ad esempio, è questo:

```
UltimaR = Cells(1, "A").End(xlDown).Row
```

Questo codice va all'ultima riga del foglio e poi risale fino a trovare la prima cella nella colonna A che contiene qualcosa. Una tecnica simile è usata anche per trovare l'ultima colonna con i dati, basta sostituire *Row* con *Column*. A questo punto possiamo aprire il file di testo in scrittura, in questo modo:

```
Open percorso For Output As #1
```

Questo comando apre un file in modalità di scrittura e un file non esiste nel percorso indicato verrà creato e se il file esiste verrà sovrascritto. Attenzione, che se si desidera che i nuovi contenuti vengano aggiunti alla fine del file, allora si usa la parola chiave **Append** al posto di **Output**. Per scrivere nel file si deve inserire la seguente riga di codice:

```
Write #1, contenutoF
```

Si deve digitare la parola chiave **Write** seguita dal numero del file e subito dopo si inserisce una virgola seguita dall'array *contenutoF* che contiene i dati che si desiderano scrivere sul file. A questo punto possiamo vedere quale codice possiamo utilizzare, innanzi tutto si crea una nuova Sub e la chiamiamo *ScriviFile* e aggiungiamo 4 variabili ad inizio routine:

```
Dim percorso As String
Dim CellaD As String
Dim UltimaC As Long
Dim UltimaR As Long
```

Per ottenere l'ultima riga e colonna con i dati, aggiungiamo le seguenti due righe:

```
UltimaR = Cells(1, "A").End(xlDown).Row
UltimaC = Cells(1, "A").End(xlToRight).Column
```

È quindi necessario azzerare la variabile CellD e specificare il percorso del file

```
CellD = ""
percorso = Application.DefaultFilePath & "\Prova.csv"
```

Oppure se sappiamo dove si trova il file possiamo utilizzare il metodo già visto per la lettura del file

```
percorso = "C:\Users\user\Documents\Test\Prova.txt"
```

Come possiamo vedere il codice punta a un file chiamato Prova.txt nella cartella Documents, si tenga presente che se non c'è tale file, VBA allora lo creerà. La riga successiva da aggiungere è quella che apre il file in scrittura:

```
Open percorso For Output As #2
```

Si noti che il numero del file alla fine è il numero 2, se ricordate abbiamo già usato il numero 1 in precedenza, quindi ora cercheremo il numero 2 per evitare eventuali conflitti.

NB *Su alcuni sistemi, si può ottenere un errore che indica che il file è già aperto quando si esegue il codice, se è così, chiudete Excel e riapritelo.*

Il codice successivo da aggiungere è il doppio ciclo For per leggere i dati e prepararli per la scrittura nel file. Questo spezzone di codice è abbastanza complesso, quindi vediamo prima il listato e poi lo commenteremo.

```
For I = 1 To UltimaR
For j = 1 To UltimaC
If j = UltimaC Then
CellD = CellD + Trim(ActiveCell(I, j).Value)
Else
CellD = CellD + Trim(ActiveCell(I, j).Value) + ","
End If
Next j
Write #2, CellD
CellD = ""
Next i
```

Come abbiamo detto, stiamo eseguendo un ciclo sulle celle del foglio di calcolo, il ciclo esterno si occupa delle righe e il ciclo interno si occupa delle colonne, inoltre all'interno del ciclo interno, abbiamo utilizzato una condizione:

```
If j = UltimaC Then
CellD = CellD + Trim(ActiveCell(I, j).Value)
Else
CellD = CellD + Trim(ActiveCell(I, j).Value) + ","
End If
```

Il motivo per cui vogliamo sapere se **j** è uguale alla variabile *UltimaC* è a causa delle virgole, in quanto vogliamo che ogni riga nel nostro file di testo abbia i dati separati in questo modo:

Mario, Rossi, 12345678

In pratica stiamo scorrendo una cella alla volta del foglio di lavoro e ognuno degli elementi deve essere separato da una virgola, tuttavia, non c'è la virgola alla fine del terzo elemento, per cui se j è uguale a UltimaC non la aggiungiamo eseguendo questo codice

```
CellD = CellD + Trim(ActiveCell(I, j).Value)
```

In pratica qualunque sia la cella attiva (ActiveCell) avrà il suo valore inserito nella variabile CellD. Tuttavia, se j non è uguale a UltimaR allora viene eseguito, il codice

```
CellD = CellD + Trim(ActiveCell(I, j).Value) + ", "
```

L'unica differenza è la virgola alla fine, al primo ciclo interno, *CellD* conterrà il valore *12345678*, mentre al secondo ciclo vi aggiungerà *12345678, Mario,* e all'ultimo ciclo aggiungerà *12345678, Mario, Rossi*. Fuori dal ciclo interno, ma poco prima di Next, abbiamo queste due righe:

```
Write #2, CellD
CellD = “”
```

La prima riga è quella che scrive in realtà la nuova linea nel file di testo, mentre la seconda riporta la variabile CellD pari a una stringa vuota. Le ultime due righe di codice chiudono il file e stampano il messaggio Fatto

```
Close # 2
MsgBox (“Fatto”)
```

Tutto il codice è simile al seguente:

```
Sub ScriviFile()
Dim percorso As String, CellaD As String, UltimaC As Long, UltimaR As Long
UltimaR = Cells(1, “A”).End(xlDown).Row
UltimaC = Cells(1, “A”).End(xlToRight).Column
    percorso = “C:\Users\user\Documents\Test\Prova.txt”
    CellData = “”

Open percorso For Output As #2
For i = 1 To UltimaR
    For j = 1 To UltimaC
        If j = UltimaC Then
        CellD = CellD + Trim(ActiveCell(i, j).Value)
    Else
        CellD = CellD + Trim(ActiveCell(i, j).Value) + “,”
        End If
    Next j
    Write #2, CellD
    CellD = “”
Next i
Close #2
MsgBox (“Fatto”)
End Sub
```

Provate ad eseguire il codice e una volta individuato il nuovo file di testo, si dovrebbe trovare una cosa che assomiglia a questa:

Fig. 4

Non preoccupatevi per i doppi apici che VBA ha aggiunto all'inizio e alla fine di ogni riga

Introduzione alle collezioni in VBA

Le **Collection**, (o Collezioni) sono una serie di elementi in cui ogni elemento ha le stesse caratteristiche, in altre parole, tutti gli elementi possono essere descritti nello stesso modo. In programmazione, una collezione è una serie di elementi in cui tutti gli elementi condividono le stesse proprietà e metodi, se presenti. VBA fornisce un oggetto Collection che è possibile utilizzare per memorizzare oggetti e dati e dispone di quattro proprietà:

- Add
- Count
- Item
- Remove

Non ci sono restrizioni sul tipo di dati che possono essere memorizzati in un oggetto Collection, e oggetti con tipi di dati diversi possono essere memorizzati nello stesso oggetto Collection. Excel dispone di molte collezioni incorporate, per esempio una cartella di lavoro ha una collezione di fogli, un foglio di lavoro ha un insieme di celle e così via. Possiamo fare riferimento a un foglio di lavoro con il suo indice nella collezione. Per esempio:

```
ActiveWorkbook.Worksheets (1).Visible
```

Oppure se il foglio è stato nominato, è possibile anche fare riferimento col nome.

```
ActiveWorkbook.Worksheets ("Dati"). Visible
```

Un altro aspetto delle Collection è che sono simili agli array, ma molto spesso viene preferita la collezione rispetto alla matrice. La ragione per cui si desidera lavorare con le collezioni piuttosto che con gli array è che si vuole evitare di dover ridimensionare le matrici ogni volta. Gli array sono fondamentalmente utile solo a patto che si sa, in anticipo, le dimensioni della nostra raccolta dei dati.

Le collezioni e gli array sono entrambi utilizzati con delle variabili di gruppo ed entrambi memorizzano una serie di oggetti simili, ad esempio un elenco di nomi o di paesi, ed è possibile manipolare facilmente e rapidamente un gran numero di elementi. Vediamo ora molto brevemente la differenza tra una variabile normale e un array. Se dobbiamo memorizzare i dati di uno studente si può facilmente farlo con una singola variabile in questo modo

```
Dim stud As Long
Stud = sheets("Foglio1").Range("A1")
```

Tuttavia si avrà da affrontare la manipolazione di dati per più studenti, pertanto immaginate di voler archiviare i dati di 100 studenti, se non è stata utilizzata una collezione o un array si avrebbe bisogno di creare un centinaio di variabili, una variabile per ogni studente. Inoltre un altro problema è che si devono utilizzare queste variabili singolarmente, cioè se si desidera memorizzare 100 valori allora abbiamo bisogno di una riga di codice ogni volta che si desidera memorizzare un valore in una variabile.

```
Dim stud1 As Long
Dim stud2 As Long
Dim stud3 As Long
stud1 = sheets("Foglio1").Range ("A1")
stud2 = sheets("Foglio1").Range ("A2")
stud3 = sheets("Foglio1").Range ("A3")
```

Come si può vedere nell'esempio sopra, la scrittura di codice come questo significherebbe dover scrivere centinaia di righe di codice ripetitivo, mentre se si utilizza una raccolta o un array è solo necessario dichiarare una variabile e utilizzare un ciclo per scorrerne tutti gli elementi, per cui con una raccolta è sufficiente una riga per poter leggere ulteriori valori. Se riscriviamo l'esempio precedente utilizzando una collezione, abbiamo solo bisogno di un paio di righe di codice

```
'crea la collezione
Dim Cstud As New Collection
'Leggi 100 valori della raccolta
Dim Col1 As Range
For Each Col1 In Foglio1.Range("A1:A100")
Cstud.Add Col1.Value
Next Col1
```

Con questo esempio abbiamo visto quello che le collezioni e gli array hanno in comune, allora, qual è la differenza e perché usare uno al posto dell'altro? La differenza principale è che con una serie normalmente si impostano le dimensioni una sola volta, questo significa che si conosce la dimensione prima di iniziare ad aggiungere elementi. Mi spiego con un esempio. Immaginate di avere un foglio di lavoro con un elenco di studenti e uno studente per riga

Cognome	*Nome*	*Materia*	*Nazione*
Abate	Carmine	Storia	Italia
Aiello	Michele	Lettere	Spagna
Aliberti	Filly	Lingue	Francia
Apostolico	Maria	Storia	Italia
Apreda	Daniele	Lettere	Germania
Boccardo	Tina	Storia	Grecia
Caldieri	Giuseppina	Matematica	Portogallo
Cantiello	Lara	Storia	Spagna
Carrella	Biagio	Lingue	Spagna
Cerrato	Maurizio	Scienze	Italia
Colombo	Verusca	Matematica	Germania
Cosenza	Biagio	Informatica	Inghilterra
Cretella	Giuseppe	Fisica	Grecia
D'amaro	Michele	Filosofia	Russia
De Crescenzo	Ilaria	Informatica	Polonia

Fig. 1

Ora se si desidera memorizzare le informazioni di ogni studente, in questo esempio si può facilmente

contare il numero di righe per ottenere il numero di studenti, in altre parole si conosce il numero di elementi in anticipo.

```
'trova l'ultima riga
Dim stud1 As Long
stud1 = Sheets("Foglio1").Range("A" & Rows.Count).End(xlUp).Row
'Crea un array di dimensione corretta
Dim arr() As Long
ReDim arr(1 To stud1)
```

Nel codice di esempio, si può vedere che si ottiene il numero di studenti contando le righe, per cui possiamo quindi utilizzare questo sistema per creare un array di dimensioni corrette. Vediamo ora in un secondo esempio in cui non conosciamo il numero di elementi in anticipo e vogliamo estrarre solo gli studenti con un dato criterio. Ad esempio solo gli studenti provenienti dalla Spagna o dall'Italia che studiano la matematica o la storia, in altre parole non sarà come selezionare uno studente e leggere i loro dati dal foglio di lavoro.

Immaginate anche che gli studenti possono essere aggiunti o rimossi dalla lista, quindi il numero di studenti non è fisso e varia, e non si conosce il numero di studenti in anticipo, quindi non sappiamo che dimensione assegnare alla matrice per crearla. Si potrebbe creare un array di grandi dimensioni, ma si avrebbero un sacco di slot vuoti, oppure leggere 50 studenti da un massimo di 1.000, ma allora si avrebbe 950 slot di matrice non utilizzati. Si potrebbe anche ridimensionare la matrice per ogni elemento come viene aggiunto, ma questo metodo è molto inefficiente e piuttosto disordinato, quindi, possiamo utilizzare una Collection in questo modo.

```
Dim coll As New Collection
coll.Add "Gino"
coll.Add "Mauro"
coll.Remove 1
```

Quando si aggiunge o si rimuove un elemento in una collezione, VBA fa tutto il ridimensionamento da solo, **non è necessario specificare le dimensioni o allocare nuovi spazi**, tutto quello che si deve fare è aggiungere un elemento o rimuoverlo. Le raccolte sono molto più facili da usare rispetto agli array soprattutto per chi non ha molta pratica nella programmazione, il più delle volte si fanno tre cose con le collezioni: Si creano, si aggiungono elementi e si leggono le voci

Si deve però tener presente nella scelta del metodo da adottare che le Collezioni hanno anche uno svantaggio, infatti **sono procedure in sola lettura**, pertanto si può aggiungere o rimuovere un elemento, ma non è possibile modificarne il valore. Se si ha la necessità di cambiare i valori in un gruppo di voci, allora è meglio utilizzare un array. Ora che sappiamo quando e perché utilizzare una collezione diamo un'occhiata a come usarle. È possibile dichiarare e creare in una sola riga del codice una collezione in questo modo

```
Dim coll As New Collection
```

Come si può vedere non è necessario specificare le dimensioni e una volta che la collezione è stata creata è possibile aggiungere facilmente elementi, inoltre è possibile dichiarare e quindi creare la collezione, quando se ne ha bisogno.

```
'Dichiarazione
Dim coll As Collection
'crea la raccolta
Set coll = New Collection
```

La differenza tra questi metodi è che nel primo la raccolta viene sempre creata, mentre nel secondo metodo viene creata solo quando si raggiunge la linea **Set**, così si potrebbe impostare il codice per creare solo la raccolta se una certa condizione viene soddisfatta

```
'Dichiarazione
Dim coll As Collection
'crea la raccolta se viene trovato un file
If filefound = True Then
Set coll = New Collection
End If
```

Il vantaggio di questo metodo è minimo, l'allocazione di memoria era un parametro importante tanti anni fa, quando la memoria del computer era limitata, a meno che non si stia creando un enorme numero di collezioni su un PC lento non si noterà alcun beneficio. Utilizzando la parola chiave **Set**, la raccolta si comporterà in modo diverso rispetto a quando si imposta la raccolta a Nothing, questa istruzione viene usata per rimuovere tutti gli elementi

```
Set coll = Nothing
```

Un punto importante da capire è che ciò che fa una collezione dipende da come è stata creata, come abbiamo visto è possibile creare una collezione dichiarandola utilizzando **New** o utilizzando **Set** e **New**. Diamo ora un'occhiata a entrambi i tipi

Creare una raccolta utilizzando New

Quando si aggiunge un nuovo elemento VBA imposta automaticamente la variabile Collection a una raccolta valida, in altre parole se si imposta la raccolta a **Nothing** vengono svuotati tutti gli elementi e se poi si aggiunge un elemento si avrà una collezione con una sola voce. Ciò rende più semplice per svuotare una raccolta, il codice seguente illustra questo aspetto

```
Sub Test ()
'crea la raccolta e aggiungi degli elementi
Dim coll As New Collection
Set coll = Nothing
Coll.Add "Pippo"
End Sub
```

Creare una raccolta utilizzando Set e New

Quando si utilizza la parola chiave **Set** per creare una collezione è necessario creare nuovamente la raccolta, se successivamente viene impostata a Nothing, nel codice che segue, dopo aver impostato la collezione a **Nothing**, si deve poi impostarla di nuovo utilizzando la parola chiave **New**, se non si esegue questa operazione si ottiene l'errore: "Variabile o Oggetto non impostato".

```
Sub Test ()
'crea la collezione
Dim coll As Collection
Set coll = New Collection
Coll.Add "Pippo"

Set coll = Nothing
Set coll = New Collection
Coll.Add "Franco"
End Sub
```

Rimuovere tutta la Collezione - un metodo alternativo

Il seguente metodo rimuove tutti gli elementi di una collezione, ma è molto lento, il vantaggio è che funziona indipendentemente dal modo in cui si crea la collezione.

```
Sub elimina (ByRef coll As Collection)
Dim k As Long
For k = coll.Count To 1 Step -1
Coll.Remove K
Next k
End Sub
```

Aggiunta di elementi a una raccolta

Per aggiungere elementi a una Collezione è possibile farlo utilizzando la proprietà **Add** seguita dal valore che si desidera aggiungere.

```
coll.Add "Pera"
coll.Add "Mela"
```

Quando si aggiungono elementi in questo modo vengono aggiunti al successivo indice disponibile, nell'esempio sopra riportato, Pera si aggiunge alla posizione 1 e Mela alla posizione 2.

I Parametri Before e After

È possibile utilizzare i parametri **Before** e **After** per specificare dove si desidera posizionare l'elemento della collezione.

```
coll.Add "Pera"
coll.Add "Mela"
'Aggiungere Limone come prima voce
coll.Add "Limone", Before: =1
```

Dopo questo codice la collezione è nell'ordine
1. Limone
2. Pera
3. Mela

```
coll.Add "Pera"
coll.Add "Mela"
'Aggiungere Limone dopo la prima voce
coll.Add "Limone", After: =1
```

Dopo questo codice la collezione è nell'ordine
1. Pera
2. Limone
3. Mela

Accesso agli elementi di una collezione

Per accedere alle voci di una raccolta è sufficiente utilizzare l'indice, che come abbiamo già visto, è la posizione della voce nella raccolta in base all'ordine che sono stati aggiunti. L'ordine può anche essere impostato utilizzando il parametro Before o After.

```
Sub Test ()
Dim coll As New Collection
coll.Add "Pera"
coll.Add "Mela"
' stampare Pera nella finestra Immediata
Debug.Print coll (1)
' Aggiungere fragola come prima voce
coll.Add "Fragola", Before: = 1
' stampare Fragola nella finestra Immediata
Debug.Print coll (1)
' stampare Pera nella finestra Immediata che ora è nella seconda posizione
Debug.Print coll (2)
End Sub
```

È inoltre possibile utilizzare la proprietà **Item** per accedere a un elemento della collezione, che è il metodo predefinito della raccolta in modo che le linee di codice siano equivalenti

```
Debug.Print coll (1)
Debug.Print coll.Item (1)
```

Abbiamo detto che non è possibile modificare il valore di un elemento in una collezione, perché quando si accede a un elemento di una raccolta è di sola lettura, per cui se si tenta di scrivere una voce della raccolta si ottiene un errore. Il seguente codice produce un errore "Necessario oggetto"

```
Sub Test ()
Dim coll As New Collection
Coll.Add "Mela"
' questa riga genera un errore
Coll (1) = "Pera"
End Sub
```

Aggiungere tipi di dati diversi

È inoltre possibile aggiungere diversi tipi di oggetti alla collezione.

```
coll.Add "Mela"
coll.Add 45
coll.Add N° 12/12/2015 #
```

Il codice seguente visualizza il tipo e il nome di tutti i fogli della cartella di lavoro corrente.

Nota: *Per accedere a tipi di dati diversi è necessario che la variabile sia dichiarata come Variant o avrete un errore.*

Si utilizza un ciclo For Each...Next quando si desidera ripetere un set di istruzioni per ciascun elemento di una raccolta o di una matrice.

```
Sub Test ()
Dim sh As Variant
For Each sh In ThisWorkbook.Sheets
' stampa nella finestra immediate il nome del foglio e il tipo di foglio
Debug.Print TypeName (sh), sh.Name
Next sh
End Sub
```

Aggiunta di elementi utilizzando una chiave (Key)

È inoltre possibile aggiungere elementi utilizzando una chiave come l'esempio sotto riportato

```
Coll.Add Item:=45, Key:="Gino"
Debug.Print "Hai inserito: ",coll("Bill")
```

Ho incluso i nomi dei parametri per rendere l'esempio più chiaro, tuttavia non è necessario, basta ricordare la chiave, che è il secondo parametro, e deve essere una stringa univoca. Il codice seguente mostra un secondo esempio di utilizzo di chiavi

```
Sub Test()
Dim coll1 As New Collection
    coll1.Add 45, "Gino"
    coll1.Add 67, "Franco"
    coll1.Add 12, "Laura"
    coll1.Add 89, "Bruno"
    Debug.Print coll1("Franco")
    Debug.Print coll1("Bruno")
End Sub
```

Utilizzando la chiave si hanno tre vantaggi:

- Se l'ordine cambia il codice può accedere lo stesso alla voce corretta
- È possibile accedere direttamente alla voce senza leggere l'intera collezione
- Il codice è più leggibile

Nella collezione VBA delle cartelle di lavoro è molto meglio accedere alla cartella di lavoro con la chiave (il nome) in quanto l'indice è poco affidabile perché dipende da quando sono stati aperti e quindi è molto casuale.

Quando utilizzare le chiavi

Un esempio di quando usare le chiavi è il seguente: Immaginate di avere una collezione di nomi di 10.000 studenti con i loro ID (indici), si potrebbe aggiungere i 10.000 studenti ad una collezione utilizzando il loro ID studente come chiave. Quando si legge un ID dal foglio di lavoro è possibile accedere direttamente al nome dello studente.

Problemi ad usare le chiavi nelle collezioni

Ci sono tre problemi con l'utilizzo delle chiavi nelle collezioni

- Non è possibile controllare se la chiave esiste
- Non è possibile modificare la chiave
- Non è possibile recuperare la chiave

VBA contiene una classe simile alla Collezione chiamata dizionario, che permette di utilizzare sempre le chiavi per aggiungere un elemento. Il dizionario fornisce ulteriori funzionalità per lavorare con le chiavi e se avete bisogno di più funzionalità con le chiavi, si potrebbe trovare il Dizionario molto utile. Tornando alle Collezioni, se si ha bisogno di accedere direttamente a un singolo elemento possono essere molto utili, in caso contrario, non è conveniente usarle.

Accedere a tutti gli elementi di una collezione

Per accedere a tutti gli elementi di una collezione è possibile utilizzare un ciclo For o un For Each. Diamo un'occhiata a questi singolarmente.

Utilizzo del ciclo For

Con un normale ciclo For, si utilizza l'indice per accedere a ciascuna voce. L'esempio seguente stampa il nome di tutte le cartelle di lavoro aperte

```
Sub Test()
Dim k As Long
For k = 1 To Workbooks.Count
Debug.Print Workbooks(k).Name
Next k
End Sub
```

Come si può vedere nel codice abbiamo usato come indice 1 con Workbooks.Count, in quanto il primo elemento è sempre in posizione 1 e l'ultimo elemento è sempre nella posizione specificata dalla proprietà Count dell'insieme. Il prossimo esempio stampa tutti gli elementi di una collezione creati dagli utenti.

```
Sub Test()
'Dichiarare e creare la raccolta
    Dim coll2 As New Collection
    ' aggiungere gli articoli
    coll2.Add "Mela"
    coll2.Add "Pera"
    coll2.Add "Fragola"
    ' stampare tutte le voci
    Dim i As Long
    For i = 1 To coll2.Count
        Debug.Print coll2(i)
    Next i
End Sub
```

Utilizzo del ciclo For Each

Il ciclo For Each non usa l'indice ed il formato è mostrato nel seguente esempio

```
Sub Test()
Dim sh As Variant
    For Each sh In Workbooks
        Debug.Print sh.Name
    Next
End Sub
```

Il formato del **ciclo For** è:
For i = 1 To coll2.Count
Next i
dove i è una variabile Long e coll2 è una collezione. Il formato del **ciclo For Each** è:
For Each var In coll2
Next
dove var è una variabile Variant e coll2 è una raccolta. È importante comprendere la differenza tra i due cicli, il ciclo For Each:

- E' più veloce
- E' più ordinato da scrivere
- Ha un solo ordine, dal più basso al più alto

Mentre invece Il Ciclo For

- E' più lento
- E' meno ordinato da scrivere
- Si può accedervi con un ordine diverso

Se mettiamo a confronto i due cicli il For Each è considerato più veloce rispetto al ciclo For ed è più ordinato da scrivere, soprattutto se si utilizzano cicli annidati e si ha meno probabilità di avere errori. L'ordine del ciclo For Each è sempre dall' indice più basso al più alto e se si vuole ottenere un ordine diverso, allora si deve utilizzare il ciclo For, dove l'ordine può essere modificato. Inoltre è possibile leggere le voci in senso inverso, leggere una sezione degli articoli o è possibile leggere ogni due fogli.

```
Sub Test()
Dim i As Long 'scorrere i fogli da destra a sinistra
For i = ThisWorkbook.Worksheets.Count To 1 Step -1
Debug.Print ThisWorkbook.Worksheets(i).Name
Next i
For i = 1 To 3 'scorrere i primi 3 fogli
Debug.Print ThisWorkbook.Worksheets(i).Name
Next i
For i = 1 To ThisWorkbook.Worksheets.Count Step 2 'scorrere ogni due fogli
Debug.Print ThisWorkbook.Worksheets(i).Name
Next i
End Sub
```

Le collezioni sono una parte molto utile di VBA, sono più facili da usare rispetto agli Array e sono molto utili quando il numero di elementi cambia. Hanno solo quattro proprietà: **Add, Remove, Count e Item** e questo li rende molto facile da padroneggiare.

Introduzione alle Istruzioni condizionali

In alcuni compiti di programmazione, è necessario scoprire se una data situazione porta un valore valido e questa operazione viene fatta controllando una condizione. A supporto di questo, il linguaggio Visual Basic fornisce una serie di parole chiave e operatori che possono essere combinati per eseguire questo controllo, verificare se una condizione produce un risultato vero o falso.

Una volta che la condizione è stata verificata, è possibile utilizzare il risultato (True o False) per compiere delle azioni. Ci sono diversi modi per controllare una condizione, anche utilizzando diversi tipi di parole chiave per verificare cose diverse, essendo ben consapevoli di ciò che ciascuna parola chiave fa o non può fare per è importante scegliere quella giusta. La dichiarazione IF ... Then esamina la veridicità di un'espressione ed è strutturata in questo modo:

If CondizioneX Then Dichiarazione

Pertanto, il programma esamina una condizione, in questo caso *CondizioneX* che può essere una semplice espressione o una combinazione di espressioni e se CondizioneX è vera, allora il programma esegue la dichiarazione. Ci sono due modi per utilizzare la dichiarazione IF ... Then, se la formula condizionale è abbastanza breve, un modo è quello di scrivere su una riga, in questo modo:

```
Sub Test()
    Dim ricco As Boolean, tassa As Double
    tassa = 30
    MsgBox ("Imposta : " & tassa & "%")
    ricco = True

    If ricco = True Then tassa = 40.5
    MsgBox ("Imposta : " & tassa & "%")
End Sub
```

Questo listato produrrebbe:

Fig. 1

Fig. 2

Se per verificare la condizione ci sono molte istruzioni da eseguire, si dovrebbero scrivere le istruzioni su righe alternate, naturalmente, è possibile utilizzare questa tecnica anche se la condizione che si sta esaminando è breve. Se si scrive l'istruzione condizionale in più di una riga, è necessario terminare con *End IF* su una riga propria. La formula utilizzata è:

```
If CondizioneX Then
    Dichiarazione
End If
```

Ecco un esempio:

```
Sub Test()
    Dim ricco As Boolean, tassa As Double
    tassa = 30
    MsgBox ("Imposta : " & tassa & "%")
    ricco = True

    If ricco = True Then
    tassa = 40.5
    MsgBox ("Imposta : " & tassa & "%")
    End If
End Sub
```

E' inoltre possibile utilizzare il valore predefinito di una espressione booleana, infatti abbiamo visto che quando si dichiara una variabile booleana, per impostazione predefinita, viene inizializzata con il valore False. Ecco un esempio:

```
Sub Test()
    Dim ricco As Boolean
    MsgBox ("Sei ricco? " & ricco)
 End Sub
```

Questo produrrebbe:

Fig. 3

Sulla base di questo, se si desidera controllare se una variabile booleana dichiarata di recente e non inizializzata è uguale a False, è possibile omettere l'espressione di *= False*. Ecco un esempio:

```
Sub Test()
    Dim ricco As Boolean, tassa As Double
    tassa = 33
    If ricco Then tassa = 40.5
    MsgBox ("Percentuale tassabile : " & tassa & "%")
End Sub
```

Questo produrrebbe:

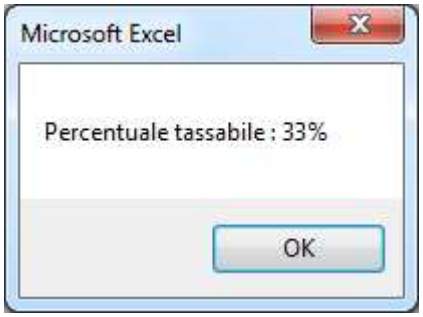


Fig. 4

Si noti che non ci sono simboli di uguale **(=)** dopo l'espressione *If ricco*, in questo caso, il valore della variabile è False, d'altra parte, se si vuole verificare se la variabile è True (Vera), assicuratevi di includere l'espressione = True. In generale, in caso di dubbio, è più sicuro inizializzare sempre la variabile e includere l'espressione = True o = False quando si valuta la variabile:

```
Sub Test()
    Dim ricco As Boolean, tassa As Double
    tassa = 33
    ricco = True
    If ricco = False Then tassa = 40.5
    MsgBox ("Percentuale tassabile : " & tassa & "%")
End Sub
```

Va ricordato che alcune funzioni booleane come *IsNumeric* e *IsDate*, il loro valore di default è True, questo significa che quando vengono richiamate è possibile omettere l'espressione = True. E' bene ricordare che quando una condizione è True o False la dichiarazione IF .. Then offre una sola alternativa: agire se la condizione è vera ed ogni volta che si desidera applicare un'espressione alternativa nel caso in cui la condizione è False, è possibile utilizzare la dichiarazione If .. Then .. Else. La formula di questa affermazione è:

```
If CondizioneX Then
    Dichiarazione1
Else
    Dichiarazione2
End If
```

Quando viene eseguita questa sezione di codice, se CondizioneX è True (Vera), allora viene eseguita l'istruzione contenuta in Dichiarazione1, se invece CondizioneX è False (Falsa), viene eseguita l'istruzione contenuta in Dichiarazione2. Ecco un esempio:

```
Sub Test()
    Dim eta As Integer, FasciaEta As String
    eta = 16
    If eta <= 18 Then
        FasciaEta = "Ragazzo"
    Else
        FasciaEta = "Adulto"
    End If
    MsgBox ("Categoria : " & FasciaEta)
End Sub
```

Ciò produrrebbe:

Fig. 5

IF Immediato

Per aiutarvi con il controllo di una condizione e la sua alternativa, il linguaggio Visual Basic fornisce una funzione chiamata **IIf** . La sua sintassi è:

```
Public Function IIf( _
   ByVal Expression As Boolean, _
   ByVal TruePart As Variant, _
   ByVal FalsePart As Variant _
) As Variant
```

Questa funzione opera come una condizione IF ... Then ... Else, prende tre argomenti richiesti e restituisce un risultato di tipo Variant e il valore restituito conterrà il risultato della funzione. Si deve tenere presente che la condizione da controllare viene passata come primo argomento alla funzione e se tale condizione è vera, la funzione restituisce il valore Vero e l'ultimo argomento viene ignorato, mentre se la condizione è falsa, il primo argomento viene ignorato e la funzione restituisce il valore del secondo argomento. Come già menzionato, è possibile recuperare il valore dell'argomento giusto e assegnarlo al risultato della funzione e il listato che abbiamo visto poco sopra può essere scritto come segue:

```
Sub Test()
    Dim eta As Integer, FasciaE As String
    eta = 16
    FasciaE = IIf(eta <= 18, "Ragazzo", "Adulto")
    MsgBox ("Categoria : " & FasciaE)
End Sub
```

Ciò produrrebbe lo stesso risultato che abbiamo visto in precedenza.

Scelta di un valore

Abbiamo imparato come controllare se una condizione è vera o falsa ed eseguire un'azione. Ecco un esempio:

```
Sub Test()
    Dim flag As Integer, tipo As String
    flag = 1
    tipo = "Sconosciuto"
    If flag = 1 Then
        tipo = "Tempo Pieno"
    End If
    MsgBox ("Tipo di impiego : " & tipo)
End Sub
```

Per fornire un'alternativa a questa operazione, il linguaggio Visual Basic fornisce una funzione chiamata **Choose**, la cui sintassi è:

```
Public Function Choose( _
   ByVal Index As Double, _
   ByVal ParamArray Choice() As Variant _
) As Variant
```

Questa funzione richiede due argomenti richiesti, il primo è equivalente alla CondizioneX della formula If ... Then. Per la funzione Choose, questo primo argomento deve essere un numero ed è il valore rispetto al quale viene confrontato il secondo argomento. Prima di richiamare la funzione, è necessario conoscere il valore del primo argomento ed è possibile farlo dichiarando prima una variabile e inizializzarla con il valore desiderato. Ecco un esempio:

```
Sub Test()
    Dim flag As Byte, tipo As String
    flag = 1
    tipo = Choose(flag, ...)
    MsgBox ("Tipo di impiego : " & tipo)
End Sub
```

Il secondo argomento può essere la dichiarazione della nostra formula. Ecco un esempio:

Choose(flag, "Tempo Pieno")

Il secondo argomento è in realtà un elenco di valori e ogni valore ha una posizione specifica detta indice. Si tenga presente che per utilizzare la funzione in uno scenario If ... Then, si passa un solo valore come secondo argomento e questo valore (o argomento) ha un indice di 1. Quando la funzione Choose viene richiamata in un ciclo If, se il primo argomento contiene il valore 1, il secondo argomento viene convalidato. Quando la funzione Choose viene richiamata, restituisce un valore di tipo Variant ed è possibile recuperare quel valore, memorizzarlo in una variabile e usarlo all'occorrenza. Ecco un esempio:

```
Sub Test()
    Dim flag As Byte, tipo As String
    flag = 1
    tipo = Choose(flag, "Tempo Pieno")
    MsgBox ("Tipo di impiego : " & tipo)
End Sub
```

Ciò produrrebbe:

Fig. 6

In alcuni casi, la funzione Choose può produrre un risultato nullo, se consideriamo lo stesso listato usato in precedenza, ma con un diverso valore:

```
Sub Test()
    Dim flag As Byte, tipo As String
    flag = 2
    tipo = Choose(flag, "Tempo Pieno")
    MsgBox ("Tipo di impiego : " & tipo)
End Sub
```

Ciò produrrebbe un errore come in figura sotto riportata

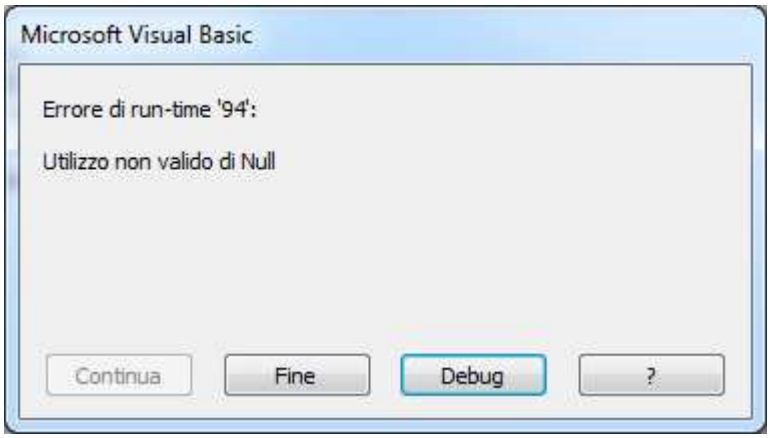


Fig. 7

In quanto non vi è alcun valore con indice 2 dopo la variabile che è stato inizializzato con il valore 2. Per utilizzare questa funzione come alternativa all'operazione If ... Then ... Else, si devono passare due valori per il secondo argomento in quanto ogni valore ha una posizione specifica in base al suo indice. Per utilizzare la funzione in una applicazione If ... Then ... Else, si devono passare due valori per il secondo argomento. Ecco un esempio:

Choose(Tipo di Impiego, "Tempo Pieno", "Part Time")

Il secondo argomento della funzione, che è il primo valore dell'argomentazione Choose, ha un indice di 1, mentre il terzo argomento della funzione, che è il secondo valore dell'argomento Choose, ha un indice di 2. In sostanza quando viene richiamata la funzione Choose, se il primo argomento ha il valore 1, il secondo argomento viene convalidato, mentre se il primo argomento ha il valore 2, viene convalidato il terzo argomento. Come già menzionato, è possibile recuperare il valore restituito della funzione, ecco un esempio:

```
Sub Test()
    Dim flag As Byte, tipo As String
    flag = 2
    tipo = Choose(flag, "Tempo Pieno")
    MsgBox ("Tipo di impiego : " & tipo)
End Sub
```

Ciò produrrebbe:

Fig. 8

Il passaggio di un valore

Come ulteriore alternativa a un If ... Then, il linguaggio Visual Basic fornisce una funzione denominata **Switch** . La sua sintassi è:

```
Public Function Switch( _
    ByVal ParamArray VarExpr() As Variant _
) As Variant
```

Questa funzione richiede un argomento obbligatorio e per utilizzarla in uno scenario If ... Then, si deve passare l'argomento come segue:

Switch(CondizioneX, Statement)

Dove CondizioneX è un segnaposto e si deve passare una espressione booleana che può essere valutata a Vero o Falso, se tale condizione è vera, il secondo argomento sarebbe stato ignorato. Quando la funzione Switch viene richiamata, si produce un valore di tipo Variant (come una stringa) che è possibile utilizzare all'occorrenza, ad esempio, è possibile memorizzarla in una variabile in questo modo:

```
Sub Test()
    Dim flag As Byte, tipo As String
    flag = 1
    tipo = "sconosciuto”
    tipo = Switch(flag = 1, "Part Time")
    MsgBox ("Tipo di impiego : " & tipo)
End Sub
```

In questo esempio, abbiamo utilizzato un numero come argomento, ma è anche possibile utilizzare un altro tipo di valore, come ad esempio una enumerazione. Ecco un esempio:

```
Private Enum tipo
    FullTime
    PartTime
    Stagionale
    Sconosciuto
End Enum

Sub Test()
    Dim flag As tipo, Result As String
    flag = tipo.FullTime
    Result = "Sconosciuto"
    Result = Switch(flag = tipo.FullTime, "Full Time")
    MsgBox ("Tipo di Impiego : " & Result)
End Sub
```

Quando si utilizza la funzione Switch, se si richiama con un valore che non è controllato dal primo argomento, la funzione genera un errore e per applicare questa funzione ad un ciclo If ... Then ... Else, è possibile richiamarla utilizzando la seguente formula:

Switch(CondizioneX1, Stato1, CondizioneX2, Stato2)

Dove il segnaposto CondizioneX1 passa una espressione booleana che può essere valutata a Vero o Falso, se tale condizione è vera, il secondo argomento sarebbe stato eseguito. Per fornire un'alternativa alla prima condizione, si può passare un'altra condizione come CondizioneX2 e se viene valutata come Vero, allora sarebbe stata eseguita l’istruzione2.

Introduzione alle Userform con VBA

Finora abbiamo visto come utilizzare delle finestre di dialogo che VBA mette a disposizione tramite le funzioni *MsgBox* e *InputBox*, sebbene queste funzioni possano garantire ad un programma un'ottima funzionalità il loro utilizzo è abbastanza limitato. Durante lo sviluppo di programmi più complessi è indispensabile usare finestre di dialogo, che permettano all'utente di selezionare più opzioni, scegliere elementi da una lista o digitare valori diversi, in pratica per incrementare la funzionalità del programma è necessario utilizzare finestre di dialogo personalizzate che ci permettano di velocizzare le nostre procedure, in questo contesto VBA ci mette a disposizione l'oggetto **Userform**(Finestre Utente) che ci permette la creazione e la manipolazione di finestre di dialogo personalizzate all'interno di programmi o procedure.

Utilizzando le *Userform*, è possibile costruire finestre personalizzate per visualizzare dati, o richiedere all'utente la digitazione di valori, utilizzando la logica che abbiamo impostato per la corretta esecuzione del programma, per esempio possiamo mostrare una finestra di dialogo, che mette a disposizione una serie di formati data, obbligando l'utente alla scelta di un solo formato tra quelli mostrati.

In sostanza le finestre di dialogo permettono al programma di interagire con l'utente in modo più "sofisticato" e forniscono uno strumento versatile per svolgere le normali funzioni di *Input* e *Output*. L'oggetto Userform è una finestra di dialogo vuota e contiene una barra del titolo e un pulsante di chiusura, aggiungendo *controlli* a un oggetto di tipo Userform è possibile personalizzare l'aspetto e la funzionalità della finestra di dialogo. Ogni oggetto Userform possiede *proprietà, metodi* e risponde ad *eventi*, tutti ereditati dall'oggetto Userform, inoltre ogni oggetto Userform incorpora un modulo nel quale l'utente può aggiungere i propri metodi e proprietà e nel quale può scrivere il codice che risponde ad eventi della finestra.

Cosa significa questo? Significa che possiamo definire evento qualsiasi cosa si verifichi all'interno della finestra di dialogo o in un suo controllo, tipici esempi di evento sono la pressione di un pulsante di comando o la selezione di una casella di controllo. Altri eventi possono includere la modifica di una casella di testo o la selezione di una lista, i clic del mouse, la pressione dei tasti e altre azioni interne attivano gli eventi. Gli oggetti utilizzati (finestre e controlli) rendono disponibile una serie di eventi, è quindi possibile scrivere procedure VBA che rispondono a questi eventi. Queste procedure vengono denominate **"Procedure di evento"** come per esempio la pressione di un pulsante di comando, la procedura di evento contiene tutte (e solo) le azioni da eseguire in relazione all'evento, altro esempio può essere la chiusura di una finestra tramite il pulsante di chiusura la procedura di evento viene eseguita in aggiunta all'azione causata dall'evento (in questo caso la chiusura della finestra di dialogo)

Come creare una UserForm

Per inserire in un file di Excel una Userform la procedura è abbastanza semplice, entriamo nell'editor del VBA (premiamo i tasti ALT+F11) e nella finestra del progetto vediamo il file xls con l'elenco dei fogli presenti al suo interno. Per creare una Userform seguiamo questo percorso **Inserisci - Userform** e di seguito ci compare una finestra come questa

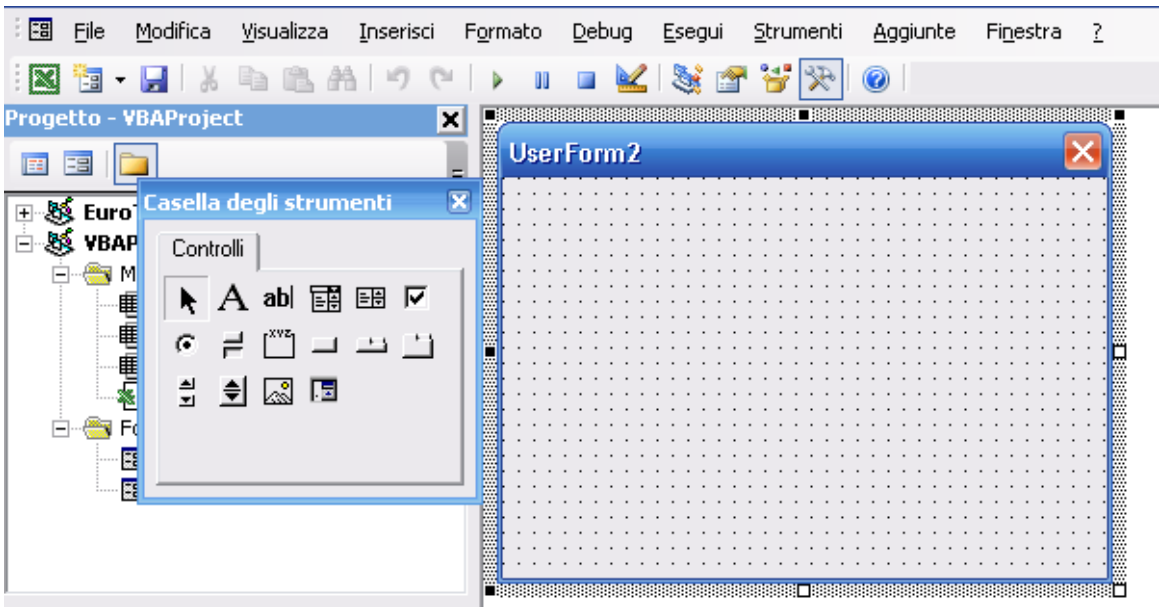


Fig. 1

Nella finestra del codice è comparsa una *UserForm* vuota e a sinistra vediamo un box denominato **Casella degli strumenti** che ci permette di inserire i vari *controlli* all'interno della Userform. La Userform ora è creata, possiamo modificarne le dimensioni a piacere, basta posizionarsi in un angolo (inferiore destro) e trascinare il mouse tenendo premuto il tasto sinistro e rilasciarlo quando abbiamo raggiunto le dimensioni desiderate.

VBA per default assegna il nome Userform seguito da un numero di indice, ma possiamo modificarne il nome anche per sapere a prima vista il compito assegnato alla Userform. Dalla finestra delle proprietà modifichiamo il nome in *Anagrafica*, facendo clic nella casella a fianco del campo (Name), cancelliamo il nome attualmente presente (Userform1) e inseriamo il nuovo.

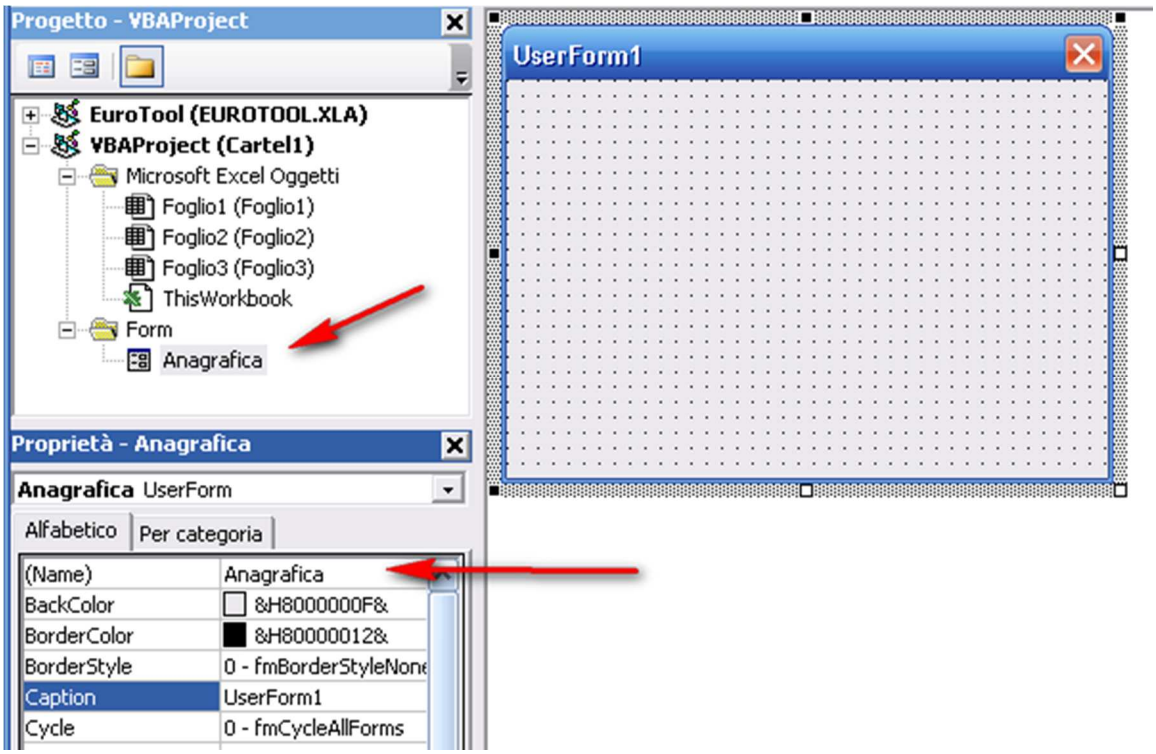


Fig. 2

Ora vediamo che nella finestra dei progetti la UserForm ha cambiato nome, ma il titolo della UserForm è rimasto inalterato, presenta ancora il nome Userform1. Per modificare il titolo sulla barra della Userform dobbiamo modificare il valore presente nel campo *caption* nella finestra delle proprietà, se modifichiamo il nome in Gestione anagrafica anche la barra della Userform assume questo titolo

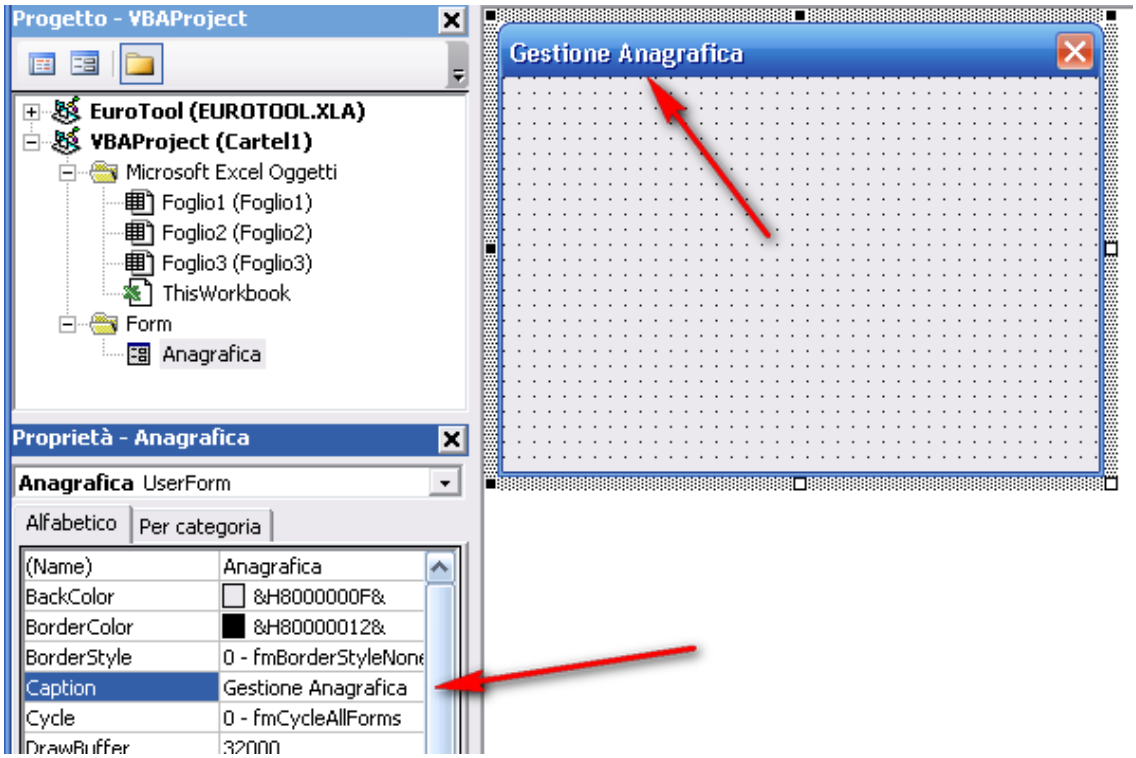


Fig. 3

Una Userform ha i suoi *eventi*, proprio come una cartella di lavoro o un foglio di lavoro ed essendo diversi, per il momento ci fermeremo a quelli più usati che sono: *Activate – Click – DblClick – Deactivate – Initialize – Terminate*
Per aggiungere eventi, dobbiamo fare doppio clic sulla finestra UserForm:

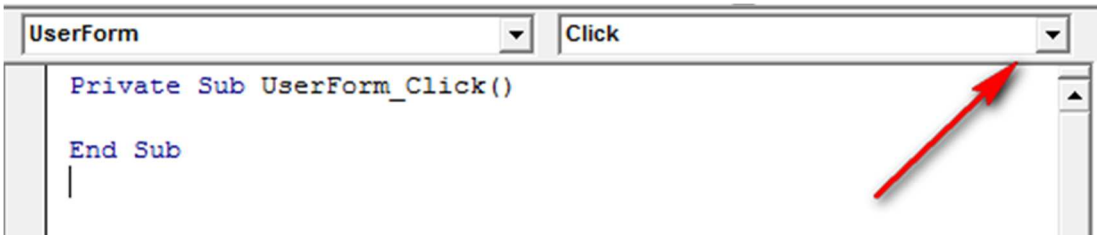


Fig. 4

Possiamo usare un evento per fissare le dimensioni della Userform, utilizzando l'evento *Userform_Initialize* che scatterà quando la form viene caricata in memoria. Questo evento può essere causato dall'istruzione *Load* o dal metodo *Show*, si utilizza questo evento per impostare l'aspetto iniziale della finestra. Per raggiungere l'evento selezioniamo il box a destra di figura 4 indicato dalla *freccia rossa* e nell'elenco a discesa scegliamo l'evento **Initialize** e inseriamo questo codice

```
Private Sub UserForm_Initialize ()
   Anagrafica.Height = 100
   Anagrafica.Width = 100
End Sub
```

Avrete notato che per agire sulla Userfom viene inserito il nome della stessa (Anagrafica) seguito dalla parola chiave Heigth (altezza) separate da un punto e con l'operatore uguale abbiamo fissato il valore a 100, la stessa operazione viene ripetuta per la larghezza con la parola chiave Width (larghezza). Se eseguiamo questa macro la Userform prenderà le dimensioni impostate, cioè sarà larga 100 pixel e alta di 100 pixel. Possiamo sfruttare un altro evento per modificare le dimensioni ogni volta che l'utente clicca sulla form aumentandole di 50 pixel ad ogni clic, inserendo questo codice nell'evento *Userform_Click*

```
Private Sub UserForm_Click ()
Anagrafica.Height = Anagrafica.Height + 50
Anagrafica.Width = anagrafica.Width + 50
End Sub
```

Sempre con lo stesso metodo che abbiamo appena visto, abbiamo inserito il nome della form seguita dal punto e dalle istruzioni Height e Width e con le assegnazioni che abbiamo visto per le variabili è stato assegnato un valore di 50 pixel preceduto dall'operatore più (+). Un altro evento che dobbiamo prendere in considerazione è l'evento *Activate* che viene scatenato ogni volta che la form diventa attiva (cioè passa in primo piano). Si utilizza questo evento per aggiornare il contenuto dei controlli, in modo da riflettere i cambiamenti che possono essere avvenuti mentre la finestra non era attiva. Possiamo usare questo evento per preparare i vari controlli a ricevere i dati dell'utente, per esempio se siamo in presenza di un TextBox(casella di testo) o vari OptionButton(pulsante di selezione) possiamo fare in modo che quando viene attivata la form il **focus** (corrisponde ad avere il campo selezionato e pronto a ricevere il testo) sia già sul Textbox oppure di avere un Optionbox già selezionato. Vediamolo con un esempio

```
Private Sub UserForm_Activate()
TextBox1.SetFocus
OptionButton1.value=True
End Sub
```

Il metodo più semplice per controllare un oggetto Userform è utilizzando i *Metodi* e le *Proprietà* predefinite della classe Userform e scrivere le procedure evento per la gestione della finestra e dei controlli in essa contenuti, i metodi più comuni da utilizzare sono: *Copy – Cut – Hide – Paste – PrintForm – Repaint – Show*

Hide : Nasconde la finestra di dialogo (la Userform) senza liberare la memoria associata all'oggetto, in questo modo vengono mantenuti i valori nei vari controlli contenuti in essa
Show : Rende visibile la finestra sullo schermo, se la finestra non è caricata in memoria, viene effettuato il caricamento.

Inoltre VBA fornisce due comandi che sono molto utili quando usiamo l'oggetto Userform, i comandi sono *Load* e *Unload*, questi comandi possono essere utilizzati per caricare l'oggetto in memoria e per liberare la memoria se occupata dall'oggetto. La sintassi per questi comandi è la seguente:

Load *Oggetto*
Unload *Oggetto*

In questo enunciato *Oggetto* rappresenta un riferimento valido ad un oggetto di tipo Userform, con il comando *Load* carichiamo l'oggetto in memoria, ma non lo rende visibile sullo schermo, e con il comando *Unload* lo scarichiamo dalla memoria.

Abbiamo appena detto che con Load carichiamo l'oggetto in memoria, ma non lo portiamo a video, inneschiamo solo l'evento *Initialize* della Userform ma per poter vedere l'oggetto sullo schermo dobbiamo usare il comando **Show**, l'enunciato è il seguente

Anagrafica.Show

Pertanto se vogliamo far comparire una finestra di dialogo a video dovremmo lanciare il comando *Show*, ma come facciamo? Una volta creata la Form abbiamo a disposizione solo metodi ed eventi, ma tutti riferiti all'oggetto Userform, mentre a noi serve un altro procedimento che veicoli la nostra finestra di dialogo. Se diamo uno sguardo alla finestra dei progetti vediamo che la Form è presente, ma abbiamo appena detto che deve essere veicolata per poter renderla visibile. Un sistema per ottenere questo è di seguire questo percorso **Inserisci - Modulo** e nella finestra di progetto ora ci comparirà anche il modulo

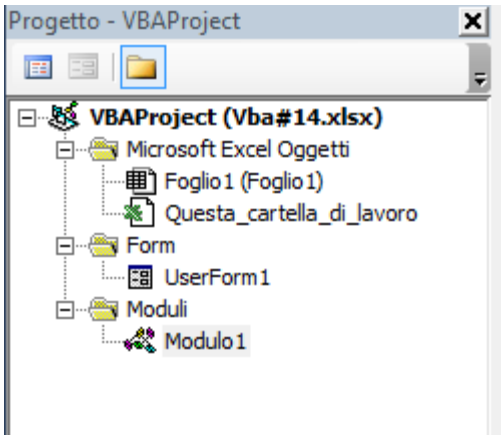


Fig. 5

A questo punto clicchiamo sulla voce "Modulo1" e nella finestra del codice digitiamo il seguente codice

```
Sub Mostra()
Anagrafica.Show
End Sub
```

Associamo ora la nostra macro ad un pulsante sul foglio di lavoro e premendo sul pulsante comparirà a video la nostra Userform. Ora abbiamo creato la nostra finestra di dialogo e siamo riusciti a portarla a video, fatte le nostre operazioni sulla form la possiamo "chiudere" usando il comando Unload. Il comando Unload và messo all'interno della Userform, associato ad un pulsante di uscita dalla stessa, la sintassi è la seguente:

Unload Me

Vediamo ora la finestra della *casella degli strumenti*, noterete che l'etichetta della finestra riporta il nome di *Controlli* e sono rappresentati dalle varie icone presenti nel box, vediamo cosa rappresentano e come si usano, per farvi comprendere meglio quali siano i controlli li ho raggruppati in questa immagine con il relativo nome e una breve descrizione

Icona	Comando	Italiano	Descrizione
	Select	Seleziona	Consente di selezionare altri controlli
	Label	Etichetta	Svolge la funzione di etichetta descrittiva non modificabile dall'utente
	TextBox	Casella di Testo	Permette di inserire del testo liberamente
	ComboBox	Casella Combinata	Permette di inserire un valore da una lista a scomparsa
	ListBox	Casella di Riepilogo	Mostra molti elementi in una lista con una barra di scorrimento verticale
	CheckBox	Casella di Controllo	Si utilizza per le opzioni che possono avere un valore tipo Vero/Falso
	OptionButton	Pulsante di Opzione	Consente all'utente di effettuare una scelta in una serie di pulsanti
	ToggleButton	Pulsante Interrutore	Mostra lo stato Vero/Falso e si presenta come un interruttore On/Off

Fig. 6

Icona	Comando	Italiano	Descrizione
	Frame	Cornice	Raggruppa una serie di controlli con un'etichetta descrittiva
	CommanButton	Pulsante di Comando	E' un pulsante standard per eseguire azioni
	TabStrip	Schede	tabulazione che può ricevere altri controlli
	Multipage	Pagine	Consiste in un'area di più pagine che può contenere altri controlli
	ScrollBar	Barra di Scorrimento	Permette di scorrere un insieme di dati
	SpinButton	Casella di Selezione	Particolare casella di testo che permette l'inserimento facilitato dei dati
	Image	Immagine	Permette di inserire un'immagine all'interno di una finestra
	RafEdit	Casella Speciale	E' una casella di testo speciale che consente di selezionare intervalli di celle

Fig. 7

Invio di una e-mail utilizzando un server remoto con CDO

Il funzionamento di un servizio di posta elettronica è garantito dalla collaborazione di tre distinti software: un server che si occupa della consegna dei messaggi (protocollo SMTP), un server che riceve i messaggi, li archivia e li trasmette, su richiesta, al titolare della casella (protocollo POP3 o IMAP) e un client che, dal PC dell'utente, è in grado di inviare e ricevere e-mail collegandosi e dialogando con le due parti server del servizio. I Web server non offrono funzionalità di posta elettronica, tuttavia le macchine che ospitano servizi Web, spesso e volentieri, permettono anche l'invio delle e-mail, tramite un apposito server SMTP.

A partire dalla versione Windows NT/2000, è stata integrata una libreria COM, chiamata **CDO**, che è l'acronimo di *Collaboration Data Object*, contenuta nel file Cdosys.dll, e permette di spedire e-mail attraverso il protocollo SMTP (Simple Mail Transfert Protocol) del Sistema Operativo, senza alcun bisogno di utilizzare un client. Vedremo come è possibile sfruttare il server SMPT presente nel sistema per inviare e-mail generate dinamicamente aiutandoci con qualche esempio.

Esempio 1: Inviare una e-mail utilizzando Gmail.
Per poter utilizzare gli oggetti della libreria CDO è innanzitutto indispensabile istanziare tali oggetti. In questo esempio useremo solo l'oggetto *CDO.Message*in cui verrà definito il messaggio stesso e per istanziare l'oggetto useremo il comando: *Set cdoMess = CreateObject("CDO.message")*. La routine completa è la seguente

```
Sub inviaE()
Set cdoMess = CreateObject("CDO.message") 'istanza dell'oggetto CDO
With cdoMess.Configuration.Fields 'Impostare i parametri di configurazione del server remoto
'Indicare come il messaggio deve essere inviato. I valori sono: 1 e 2
.Item("http://schemas.microsoft.com/cdo/configuration/sendusing") = 2
'Impostare il server SMTP che si utilizza
  .Item("http://schemas.microsoft.com/cdo/configuration/smtpserver") = "smtp.gmail.com"
'Impostare la porta di comunicazione del server SMTP
  .Item("http://schemas.microsoft.com/cdo/configuration/smptserverport") = 587
'Abilitare autenticazione SMTP, il valore 1 indica True
  .Item("http://schemas.microsoft.com/cdo/configuration/smtpauthenticate") = 1
'Abilita autenticazione SSL
  .Item("http://schemas.microsoft.com/cdo/configuration/smtpusessl") = True
'Timeout di connessione in secondi (il tempo massimo in cui CDO tenterà di stabilire una
connessione al server SMTP
  .Item("http://schemas.microsoft.com/cdo/configuration/smtpconnectiontimeout") = 60
'Impostare le credenziali del vostro account Gmail
  .Item("http://schemas.microsoft.com/cdo/configuration/sendusername") = "prova@gmail.com"
  .Item("http://schemas.microsoft.com/cdo/configuration/sendpassword") = "pass1234"
  .Update 'Aggiorna i campi di configurazione
End With
With cdoMess 'Proprietà del messaggio
'Inserire l'indirizzo del destinatario. E' possibile inserire più indirizzi separandoli con una virgola
  .To = info.prova@gmail.com
  .From = prova@gmail.com 'Inserire il nominativo del mittente della e-mail
  .Subject = "Invio email con CDO" 'Inserire l'oggetto della e-mail
'Inserire il corpo della email in testo semplice
  .TextBody = "Inserire il corpo del messaggio." & vbCRLF & "Si consiglia di creare una variabile per
contenere tutto il testo"
  .Send 'Spedizione della mail
End With
    Set cdoMess = Nothing
End Sub
```

Il codice sopra riportato è ampiamente commentato e non serve aggiungere ulteriori spiegazioni, possiamo però perfezionare il codice aggiungendo altri parametri. Per esempio, supponiamo di avere i dati di spedizione della mail in un foglio di lavoro denominato "Setup" e di doverli recuperare in automatico.

	A	B
1	**Mittente Email**	prova@gmail.com
2	**Destinatario**	info.prova@gmail.com
3	**Oggetto**	Prova invio email con CDO
4	**Messaggio**	Messaggio di prova
5	**Server SMTP**	smtp.gmail.com
6	**Password**	**********

Fig. 1

In questi casi sarebbe opportuno dividere la procedura in 2 routine, in cui una si occupa di recuperare i dati presenti nel foglio di lavoro e passarli alla seconda che li processerà e invierà l'email. Per recuperare i dati dal foglio "Setup" possiamo usare un codice come il seguente:

```
Sub Prova()
 Dim Eto As String, Efrom As String, Eogg As String, Emess As String, Esmtp As String,
Epass As String
        Eto = Sheets("setup").Range("B2")
        Efrom = Sheets("setup").Range("B1")
        Eogg = Sheets("setup").Range("B3")
        Emess = Sheets("setup").Range("B4")
        Esmtp = Sheets("setup").Range("B5")
        Epass = Sheets("setup").Range("B6")
    inviaE Eto, Efrom, Eogg, Emess, Esmtp, Epass
End Sub
```

Mentre la routine che invierà la mail presenterà questo codice

```
Sub inviaE(Eto As String, Efrom As String, Eogg As String, Emess As String, Esmtp As
String, Epass As String)
'istanza dell'oggetto CDO
Set cdoMess = CreateObject("CDO.message")
With cdoMess.Configuration.Fields
.Item("http://schemas.microsoft.com/cdo/configuration/sendusing") = 2
.Item("http://schemas.microsoft.com/cdo/configuration/smtpserver") = Esmtp
.Item("http://schemas.microsoft.com/cdo/configuration/smptserverport") = 587
.Item("http://schemas.microsoft.com/cdo/configuration/smtpauthenticate") = 1
.Item("http://schemas.microsoft.com/cdo/configuration/smtpusessl") = True
.Item("http://schemas.microsoft.com/cdo/configuration/smtpconnectiontimeout") = 60
.Item("http://schemas.microsoft.com/cdo/configuration/sendusername") = Efrom
.Item("http://schemas.microsoft.com/cdo/configuration/sendpassword") = Epass
.Update
End With
With cdoMess
.To = Eto
.From = Efrom
.Subject = Eogg
.TextBody = Emess
.Send
End With
Set cdoMess = Nothing
End Sub
```

Possiamo anche allegare uno o più file al messaggio modificando il ciclo With in questo modo:

```
With cdoMess
.To = Eto
.From = Efrom
.Subject = Eogg
.TextBody = Emess
.AddAttachment "C:\Test\info.txt"
.Send
End With
```

Oppure inserire il percorso nel foglio "Setup" e allegarlo sotto forma di variabile come visto in precedenza.
A volte può presentarsi la necessità di dover spedire delle email con un testo standard, può essere una promozione di certi prodotti, oppure un avviso qualsiasi, in questi casi è possibile inserire il testo presente in un file nel corpo della mail in questo modo:

```
'Queste costanti sono definite per rendere il codice più leggibile
Const ForReading = 1, ForWriting = 2, ForAppending = 8
Dim fso, f
Set fso = CreateObject("Scripting.FileSystemObject")
'Aprire il file in lettura
Set f = fso.OpenTextFile("C:\Test\annuncio.txt", ForReading)
'Il metodo ReadAll legge l'intero file nella variabile BodyText
BodyText = f.ReadAll
'Chiudi il file
f.Close
Set f = Nothing
Set fso = Nothing

With cdoMess
.To = Eto
.From = Efrom
.Subject = Eogg
.TextBody = BodyText
.Send
End With

Set cdoMess = Nothing
End Sub
```

Il codice sopra riportato mostra come leggere il contenuto di un file (annuncio.txt) e inserirlo in una variabile (BodyText) che costituirà il corpo del messaggio. La routine completa in questo caso diventa:

```
Sub inviaE(Eto As String, Efrom As String, Eogg As String, Emess As String, Esmtp As
String, Epass As String)
Set cdoMess = CreateObject("CDO.message") 'istanza dell'oggetto CDO

With cdoMess.Configuration.Fields
.Item("http://schemas.microsoft.com/cdo/configuration/sendusing") = 2
.Item("http://schemas.microsoft.com/cdo/configuration/smtpserver") = Esmtp
.Item("http://schemas.microsoft.com/cdo/configuration/smptserverport") = 587
.Item("http://schemas.microsoft.com/cdo/configuration/smtpauthenticate") = 1
.Item("http://schemas.microsoft.com/cdo/configuration/smtpusessl") = True
.Item("http://schemas.microsoft.com/cdo/configuration/smtpconnectiontimeout") = 60
.Item("http://schemas.microsoft.com/cdo/configuration/sendusername") = Efrom
.Item("http://schemas.microsoft.com/cdo/configuration/sendpassword") = Epass
.Update
End With

'Queste costanti sono definite per rendere il codice più leggibile
Const ForReading = 1, ForWriting = 2, ForAppending = 8
Dim fso, f
Set fso = CreateObject("Scripting.FileSystemObject")
'Aprire il file in lettura
Set f = fso.OpenTextFile("C:\Test\annuncio.txt", ForReading)
'Il metodo ReadAll legge l'intero file nella variabile BodyText
BodyText = f.ReadAll
'Chiudi il file
f.Close
Set f = Nothing
Set fso = Nothing

With cdoMess
.To = Eto
.From = Efrom
.Subject = Eogg
.TextBody = BodyText
.Send
End With

Set cdoMess = Nothing
End Sub

Sub Prova()
   Dim Eto As String, Efrom As String, Eogg As String, Emess As String, Esmtp As String,
Epass As String

      Eto = Sheets("setup").Range("B2")
      Efrom = Sheets("setup").Range("B1")
      Eogg = Sheets("setup").Range("B3")
      Emess = Sheets("setup").Range("B4")
      Esmtp = Sheets("setup").Range("B5")
      Epass = Sheets("setup").Range("B6")
   inviaE Eto, Efrom, Eogg, Emess, Esmtp, Epass

End Sub
```

A questo punto si dovrebbe aggiungere al codice una corretta gestione degli errori per evitare e prevenire errori quando la routine è in esecuzione. Possiamo modificare il listato come di seguito riportato

```
Sub inviaE(Eto As String, Efrom As String, Eogg As String, Emess As String, Esmtp As
String, Epass As String)
'istanza dell'oggetto CDO
Set cdoMess = CreateObject("CDO.message")
'Gestione errori
On Error GoTo err

With cdoMess.Configuration.Fields
.Item("http://schemas.microsoft.com/cdo/configuration/sendusing") = 2
.Item("http://schemas.microsoft.com/cdo/configuration/smtpserver") = Esmtp
.Item("http://schemas.microsoft.com/cdo/configuration/smptserverport") = 587
.Item("http://schemas.microsoft.com/cdo/configuration/smtpauthenticate") = 1
.Item("http://schemas.microsoft.com/cdo/configuration/smtpusessl") = True
.Item("http://schemas.microsoft.com/cdo/configuration/smtpconnectiontimeout") = 60
.Item("http://schemas.microsoft.com/cdo/configuration/sendusername") = Efrom
.Item("http://schemas.microsoft.com/cdo/configuration/sendpassword") = Epass
.Update
End With

'Queste costanti sono definite per rendere il codice più leggibile
Const ForReading = 1, ForWriting = 2, ForAppending = 8
Dim fso, f
Set fso = CreateObject("Scripting.FileSystemObject")
[COLOR="green"]'Aprire il file in lettura
Set f = fso.OpenTextFile("C:\Test\annuncio.txt", ForReading)
'Il metodo ReadAll legge l'intero file nella variabile BodyText
BodyText = f.ReadAll
'Chiudi il file
f.Close
Set f = Nothing
Set fso = Nothing

With cdoMess
.To = Eto
.From = Efrom
.Subject = Eogg
.TextBody = BodyText
.Send
End With

Set cdoMess = Nothing

err:
 If err.Number = 53 Then
       MsgBox "Manca il file da allegare. Procedura terminata"
       Exit Sub
    End If
End Sub
```

```
Sub Prova()

    Dim Eto As String, Efrom As String, Eogg As String, Emess As String, Esmtp As String,
Epass As String

   'controllo se esiste il destinatario
    If Trim(Sheets("setup").Range("B2")) = "" Then
        MsgBox "Manca Il Destinatario"
        Exit Sub
         Else
          Eto = Sheets("setup").Range("B2")
    End If

          'controllo se esiste il mittente
         If Trim(Sheets("setup").Range("B1")) = "" Then
           MsgBox "Manca Il Mittente"
           Exit Sub
             Else
              Efrom = Sheets("setup").Range("B1")
          End If

        'controllo è stato inserito l'oggetto della mail
         If Trim(Sheets("setup").Range("B3")) = "" Then
           MsgBox "Manca L'Oggetto della Mail"
           Exit Sub
             Else
              Eogg = Sheets("setup").Range("B3")
          End If

    Emess = Sheets("setup").Range("B4")

          'controllo se ci sono i dati dell'smtp
         If Trim(Sheets("setup").Range("B5")) = "" Then
           MsgBox "Manca SMTP da usare"
           Exit Sub
             Else
              Esmtp = Sheets("setup").Range("B5")
          End If

          'controllo se c'è la password per l'smtp
         If Trim(Sheets("setup").Range("B6")) = "" Then
           MsgBox "Manca la password dell'SMTP da usare"
           Exit Sub
             Else
              Epass = Sheets("setup").Range("B6")
          End If

    inviaE Eto, Efrom, Eogg, Emess, Esmtp, Epass

End Sub
```

La Funzione MsgBox e InputBox

Abbiamo usato questa semplice funzione molto spesso in questo corso, adesso vediamola più da vicino e come possiamo personalizzarla per i nostri progetti. La funzione **MsgBox()** ci permette di mostrare a video un box che riporterà un avviso permettendo così all'utente di scegliere l'operazione più idonea da eseguire. Questo comando ci è utile quando stiamo per mandare in esecuzione una determinata procedura e vogliamo ottenere il consenso dall'utente, oppure la possiamo usare nella gestione degli errori, ci può avvisare e impedire l'esecuzione di una routine che porterebbe il programma alla generazione di un errore con conseguente blocco dell'esecuzione del nostro progetto, ma può chiederci una ulteriore conferma per operazioni "delicate" - tipo cancellazione di file - in sostanza è una funzione che ci permette di comunicare da vicino con l'applicazione che stiamo usando.

Finora abbiamo visto marginalmente l'uso di Msgbox , in questa lezione cercheremo di approfondire le sue potenzialità e il suo uso. Non ha una sintassi particolarmente difficile ma il suo uso "avanzato" ci permette di controllare le varie procedure e gestire tutti gli eventi che abbiamo istanziato nel nostro programma, in sostanza è una funzione di VBA abbastanza semplice da usare che riporta a video un messaggio con un'icona e dei pulsanti predefiniti, a cui si risponde premendo su uno di essi. La sintassi è la seguente :

MsgBox(prompt[, buttons] [, title] [,helpfile ,context]) oppure italianizzando il commando *MsgBox(Messaggio[, Pulsanti] [, Titolo] [, Fileaiuto , Contesto])* Dove

- *Prompt o Messaggio :* indica il messaggio che sarà visualizzato nelle finestra di dialogo.
- *buttons o Pulsanti :* indica il valore numerico dei pulsanti da visualizzare nella finestra di dialogo.
- *Title o Titolo :* Indica il titolo della finestra di dialogo e va scritto fra virgolette
- *helpfile, FileAiuto* e *context o Contesto* : sono relativi alla guida dell'applicazione ma non sono indispensabili

Ricordiamoci che non abbiamo nessun controllo della posizione in cui verrà visualizzato il Box sullo schermo, vediamo ora qualche esempio :

```
Sub box1()
MsgBox "Ciao a tutti"
End Sub
```

Questo codice ci riporta a video un messaggio come questo

Fig. 1

Come potete vedere abbiamo omesso alcune espressioni nella sintassi appena esposta, ma il nostro box ci appare con il testo che abbiamo inserito, e con la barra del titolo di default [Microsoft Excel] , inoltre è possibile anche far apparire il box usando il valore di *Variabili* oppure possiamo anche usare anche una forma più evoluta e personalizzare il nostro messaggio di avviso come meglio crediamo, vediamone un esempio

```
Sub box2()
MsgBox "Ciao a tutti", vbCritical + vbOKOnly, "Funzione MsgBox Semplice"
End Sub
```

Che ci riporta a video un messaggio come questo :

Fig. 2

Notiamo subito che è cambiato il titolo nella barra della finestra ed è comparsa un'icona rossa con una X bianca, possiamo anche cambiare tipo di icona in questo modo :

```
Sub box3()
MsgBox "Ciao a tutti", vbQuestion + vbOKOnly, "Funzione MsgBox Semplice"
End Sub
```

E ci verrà riportato a video un messaggio come questo

Fig. 3

Vedendo i codici esposti e i vari box che ci sono apparsi possiamo dire che :

Prompt: è il messaggio che verrà visualizzato, nel nostro caso è *Ciao a tutti"*
buttons : è il tipo di pulsante e relativa icona, nel nostro caso è rappresentato dal codice *vbCritical + vbOKOnly*
title : è il titolo della finestra, nel nostro caso è *Funzione MsgBox Semplice"*

Ora però possiamo anche fare un altro "passo avanti" nell'uso di questa funzione, a volte è difficile ricordare tutti i comandi e potremmo usarla anche in un altro modo. Abbiamo detto poco sopra che *buttons* indica il valore numerico dei pulsanti da visualizzare nella finestra di dialogo, ma finora non abbiamo esposto valori numerici, abbiamo solo rappresentato i pulsanti con delle *"parole chiave"* tipo : *vbCritical + vbOkOnly*, oppure *vbQuestion + vbOkOnly*vediamo questo aspetto modificando il codice finora usato in questo modo :

```
Sub box4()
MsgBox "Ciao a tutti", 0 + 16, "Funzione MsgBox Avanzata" '
End Sub
```

L'esecuzione di questo codice ci riporta questo avviso

Fig. 4

Come possiamo vedere l'avviso cambia solo nel titolo che abbiamo modificato in *Funzione MsgBox Avanzata* ma il resto del Box è uguale, eppure il codice è rappresentato in maniera diversa, possiamo vedere che non compaiono più le parole chiave usate in precedenza ma abbiamo inserito dei valori numerici separati da virgole e seguiti dal titolo della finestra. A mio avviso usarla in questo modo è molto più semplice da ricordare e avremmo meno codice da scrivere. Esponiamo ora con una tabella come vengono interpretati i valori da VBA, che ci aiuterà nell'interpretazione di quanto finora esposto. L'argomento buttons indica il valore numerico dei pulsanti da visualizzare nella finestra del Box e sono così rappresentati

Valore	Descrizione Pulsanti	Comando	Icona
0	Ok	vbOKOnly	Ok
1	Ok - Annulla	vbOKCancel	Ok Annulla
2	Interrompi - Riprova - Ignora	vbAbortRetryIgnore	Interrompi Riprova Ignora
3	Si - No - Annulla	vbYesNoCancel	Si No Annulla
4	Si - No	vbYesNo	Si No
5	Riprova - Annulla	vbRetryCancel	Riprova Annulla

Fig. 5

Sempre a questo argomento possiamo associare un'icona da visualizzare nel Box identificata da un valore numerico seguendo questa tabella:

Valore	Descrizione	Comando	Icona
16	Avviso Grave	vbCritical	
32	Domanda	vbQuestion	
48	Esclamazione	vbExclamation	
64	Informazione	vbInformation	

Fig. 6

Abbiamo detto poco sopra che la funzione MsgBox *restituisce* un valore, questo valore rappresenta *il pulsante che abbiamo premuto*, infatti come facciamo a sapere quale pulsante, e di conseguenza, quale scelta ha fatto l'utente? Ora dobbiamo fare una piccola parentesi, *nelle lezioni precedenti abbiamo sempre esposto MsgBox come un semplice avviso*, invece possiamo usarla anche quando dobbiamo prendere delle decisioni, in pratica si sta dimostrando l'estrema versatilità di questa funzione, unica cosa è fondamentale sapere quale tasto è stato premuto dall'utente per consentire o negare l'esecuzione di una procedura.

A questo punto non diventa solo una semplice funzione che rimanda un avviso, ma prende campo un aspetto più importante, cioè può permettere l'esecuzione di una procedura oppure indicarci quale operazione stiamo per eseguire e richiedere un'ulteriore conferma. Vediamo ora con una tabella quali sono i valori che vengono restituiti da MsgBox e poi con qualche riga di codice ne vediamo il suo uso all'interno di una procedura

Valore	Pulsante Selezionato	Comando
1	Ok	vbOk
2	Cancel	vbCancel
3	Abort	vbAbort
4	Retry	vbRetryCancel
5	Ignore	vbIgnore
6	Yes	vbYes
7	No	vbNo

Fig. 7

In base alla tabella sopra esposta possiamo dire che se l'utente preme il tasto Ok il valore restituito sarà 1, mentre se preme il pulsante Yes il valore restituito sarà 6, di conseguenza possiamo usare la funzione MsgBox anche in altri contesti diversi dal semplice avviso, ma usarla anche per operare delle scelte ed usando le Variabili – come abbiamo detto all'inizio – per ottimizzare sia la funzione ma soprattutto l'uso che ne viene fatto. Vediamo qualche esempio di codice

```
Sub prova()
 Dim Risp As Integer
 Risp = MsgBox("Prova funzione MsgBox", 1 + 64, "Funzione MsgBox Avanzata")
  If Risp = 1 Then
  MsgBox "Hai schiacciato il pulsante Ok", 1+48, "Funzione MsgBox Avanzata"
Else
 Exit Sub
End If
End Sub
```

Con il codice sopra esposto ci compare una finestra come questa

Fig. 8

E ciccando sul pulsante Ok ci comparirà un messaggio del genere

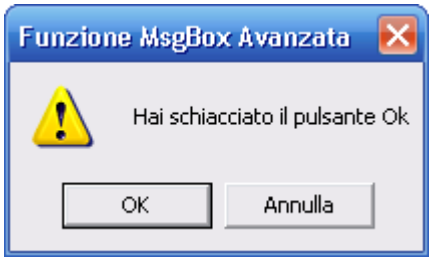


Fig. 9

Credo che sia abbastanza eloquente come solo sostituendo le condizioni da verificare e le procedure da eseguire possiamo utilizzare questa funzione con scopi ben diversi dal solo avviso, ma possiamo integrarla con le scelte che andiamo ad operare nel proseguo del nostro programma, costituendo così un'ossatura stabile e logica del nostro codice sia per quanto riguarda la gestione degli errori che richiedendo conferma all'utente di quanto si appresta a fare e in base alle scelte che effettua indirizzare il flusso del programma nella direzione appropriata

La Funzione InputBox

Se vogliamo che l'utente possa operare delle scelte su come usare la procedura possiamo usare la funzione Msgbox che abbiamo già visto nella lezione precedente oppure la funzione InputBox, la quale ci permette di ottenere un input dall'utente, la sintassi generale è la seguente:

InputBox (Messaggio) [,Titolo, Default, XPos, YPos, File Aiuto, Contesto])

Messaggio è una stringa usata per indicare all'utente quale informazione deve inserire, ed è l'unico argomento richiesto, tutti gli altri sono opzionali.
Titolo è una stringa usata come titolo per la finestra di dialogo
Default è una stringa per fornire un valore di Default per l'input dell'utente

XPos e *YPos* sono espressioni numeriche che forniscono le coordinate dove deve apparire la finestra di dialogo, *XPos* è la distanza orizzontale dal lato sinistro della finestra e *YPos* è la distanza verticale dal lato superiore della finestra, sono argomenti opzionali, ma fate attenzione se li usate perchè se specificate delle posizioni troppo grandi per questi argomenti si corre il rischio di non far apparire la finestra sullo schermo

FileAiuto è una stringa che contiene il nome di un file della guida di Windows e *Contesto* è un'espressione numerica che specifica l'argomento nel file della guida relativo alla finestra di dialogo che state visualizzando.*FileAiuto* e *Contesto* sono opzionali, ma se specificate FileAiuto dovete specificare anche Contesto, e quando specificate un file della guida per una finestra di dialogo di input, VBA aggiunge automaticamente un pulsante della Guida (?) alla finestra di dialogo. Vediamo ora un esempio

```
Sub funzione_input()
Prova_input = InputBox(prompt:="Inserisci il nome di un file: ", Title:="Crea un nuovo
file", Default:="Newfile")
End Sub
```

ed otteniamo un finestra dei questo tipo

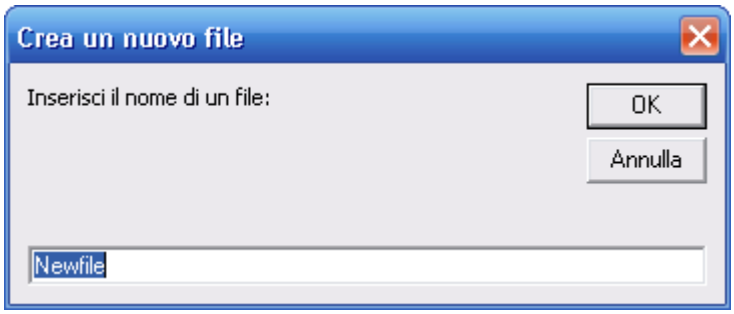


Fig. 10

credo che sia abbastanza semplice ed intuitivo il listato esposto, infatti vediamo che quando digitato nel codice appare nella finestra di dialogo, ora possiamo passare all'argomento appena accennato all'inizio cioè prendere delle decisioni, naturalmente le nostre procedure non possono veramente *"Prendere delle decisioni"* allo stesso modo di un essere umano, ma bensì possono scegliere tra diversi percorsi di azioni predefinite, basandosi su semplici condizioni e prendendo delle decisioni al solo verificarsi di determinati eventi. Possiamo dire che usiamo le istruzioni di scelta di VBA, definite in una condizione oppure in un insieme di condizioni per cui VBA esegue un blocco di codice della nostra procedura oppure un altro blocco di codice.

Lavorare con le Date in VBA

Microsoft Excel tramite Il linguaggio Visual Basic, offre una vasta gamma di funzioni per poter creare e manipolare le date ed è possibile inserire una data in qualsiasi formato leggibile, ad esempio usando lo stile americano (mese-giorno-anno) o il formato europeo (giorno-mese-anno), dipende dalle impostazioni internazionali del vostro sistema

Per memorizzare una data o un orario, e necessario definire una variabile di tipo **Date** e assegnare un valore alla variabile, con una sintassi particolare che prevede di inserire la data tra due caratteri "cancelletto" (**#**): per esempio, il 10 agosto 2015, può essere scritto:

```
Sub Prova()
    Dim Mydata As Date
   Mydata = #10/8/2015#
   MsgBox ("La data corrente è : " & Mydata)
End Sub
```

Ciò produrrebbe:

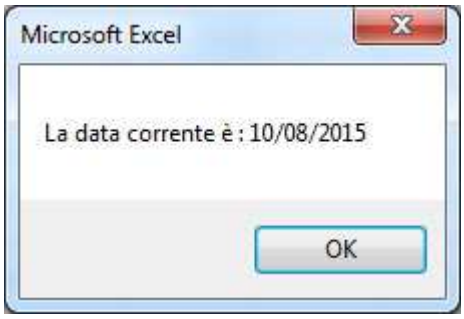


Fig. 1

Come si può notare nel codice la data è inserita nel formato americano, cioè viene espressa nella forma Mese/Giorno/Anno, perchè, le istruzioni in VBA sono in inglese, ma verrà convertita nel formato inserito nel sistema per cui nel box appare la data scritta all'italiana :10/08/2015.

La visualizzazione del formato della data dipende dalle impostazioni di settaggio contenute nel Pannello di Controllo seguendo il percorso: Pannello di controllo/Impostazioni Internazionali/Data oppure per Windows 7 o superiori dal percorso: Pannello di controllo/Paese e lingua

Per ottenere la data corrente del computer, è possibile richiamare la funzione **Date** in questo modo:

```
Sub Prova()
   MsgBox Date
End Sub
```

Quando si compone un valore data, è necessario seguire alcune regole che dipendono dalla lingua che si sta utilizzando. Vediamo quelle Italiane, dove un mese è riconosciuto da un indice in un intervallo da 1 a 12 e ha anche un nome che può essere rappresentato in due formati: *completo* o *breve*.

Indice Mese	Nome Completo	Nome Breve
1	Gennaio	Gen
2	Febbraio	Feb
3	Marzo	Mar
4	Aprile	Apr
5	Maggio	Mag
6	Giugno	Giu
7	Luglio	Lug
8	Agosto	Ago
9	Settembre	Set
10	Ottobre	Ott
11	Novembre	Nov
12	Dicembre	Dic

Fig. 2

Inoltre una settimana è una combinazione di 7 giorni consecutivi di un mese, per cui ogni giorno può essere riconosciuto da un indice da 1 a 7 e il giorno di ciascun indice è riconosciuto da un nome, dove il primo giorno ha un indice di 1 e si chiama Domenica, mentre l'ultimo giorno ha un indice di 7 si chiama Sabato. Come i mesi di un anno, i giorni di una settimana hanno nomi completi e brevi.

Indice Giorni	Nome Completo	Nome Breve
1	Domenica	Dom
2	Lunedì	Lun
3	Martedì	Mar
4	Mercoledì	Mer
5	Giovedì	Gio
6	Venerdì	Ven
7	Sabato	Sab

Fig. 3

Abbiamo visto che, durante la creazione di una data, è possibile includere il suo valore tra i segni **#**. Un'alternativa è quella di fornire una data come una stringa. A sostegno di questa, il linguaggio Visual Basic fornisce una funzione chiamata **DateValue** la cui sintassi è:

Function DateValue(ByVal StringDate As String) As Variant

Quando si chiama questa funzione, si deve fornire una data valida come argomento. La validità dipende dalla lingua del sistema operativo. Ecco un esempio:

```
Sub Prova()
    Dim data1 As Date
    data1 = DateValue("22-Ago-2010")
    MsgBox ("La Data impostata è : " & data1)
End Sub
```

Ciò produrrebbe:

Fig. 4

Un'alternativa a inizializzare una variabile data è quella di utilizzare una funzione denominata **DateSerial**, la sua sintassi è:

Function DateSerial(ByVal [Year] As Integer, ByVal [Month] As Integer, ByVal [Day] As Integer) As Variant

Come potete vedere, questa funzione consente di specificare l'anno, il mese, il giorno di un valore di data, ovviamente senza i segni **#**. Quando è stata chiamata, questa funzione restituisce un valore di tipo Variant, che può essere convertita in una data . Ecco un esempio:

```
Sub Prova()
    Dim data1 As Date
    data1 = DateSerial(2010, 2, 8)
    MsgBox ("La Data impostata è : " & data1)
End Sub
```

Ciò produrrebbe:

Fig. 5

Quando si passa il valore di questa funzione, è necessario limitare ogni componente per l'intervallo consentito di valori. È possibile passare l'anno con due cifre da 0 a 99. Ecco un esempio:

```
Sub Prova()
    Dim data1 As Date
    data1 = DateSerial(3, 2, 8)
    MsgBox ("La Data impostata è : " & data1)
End Sub
```

Se si passa l'anno come un valore compreso tra 0 e 99, l'interprete rimanda all'orologio del computer per ottenere il secolo. Al momento in cui scriviamo, il secolo sarebbe il 20 e l'anno specificato sarebbe aggiunto, che producono 2003. Per essere più precisi e ridurre la confusione, si dovrebbe sempre passare l'anno con 4 cifre.

Il mese deve essere un valore compreso tra 1 e 12, e se si passa un valore superiore a 12, l'interprete dovrebbe calcolare il resto di quel numero da 12 (il numero MOD 12 =?). Il risultato della divisione intera sarebbe stato usato come il numero di anni e aggiunto al primo argomento. Il resto sarebbe stato usato come il mese del valore data. Ad esempio, se si passa il mese come 18, la divisione intera produrrebbe 1, quindi 1 anno sarebbe stato aggiunto al primo argomento. Il resto è 6 (18 MOD 12 = 6); così il mese sarebbe stato usato come 6 (Giugno). Ecco un esempio:

```
Sub Prova()
    Dim data1 As Date
    data1 = DateSerial(2003, 18, 8)
    MsgBox ("La Data impostata è : " & data1)
End Sub
```

Ciò produrrebbe:

Fig. 6

Per fare un altro esempio, se si passa il mese come 226, la divisione intera (226 / 12) produce 18 e il numero sarebbe stato aggiunto al primo argomento (2003 + 18 = 2021). Il resto di 226 a 12 è 10 (226 MOD 12 = 10) e sarebbe stato utilizzato come il mese. Ecco un esempio:

```
Sub Prova()
    Dim data1 As Date
    data1 = DateSerial(2003, 226, 8)
    MsgBox ("La Data impostata è : " & data1)
End Sub
```

Ciò produrrebbe:

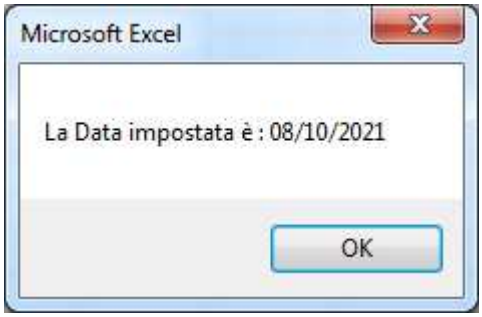


Fig. 7

Se il mese è passato come 0, si considera 12 (dicembre), mentre se è passato come -1, è considerato 11 (novembre) dell'anno precedente e così via. Se il mese è passato come un numero inferiore a -11, l'interprete avrebbe calcolare la divisione intera a 12, aggiungendo 1 a questo risultato e utilizzando tale numero come l'anno, calcolando il resto a 12, e usandolo come mese.

A seconda del mese, il valore dell'argomento giorno può essere passato come un numero compreso tra 1 e 28, tra 1 e 29, tra 1 e 30, o tra 1 e 31. Se l'argomento giorno viene passato come un numero inferiore a 1 o superiore a 31, l'interprete utilizza il primo giorno del mese passato come secondo argomento. Se il giorno è passato come -1, il giorno è considerato l'ultimo giorno del mese precedente del mese in discussione. Ad esempio, se l'argomento mese è passato come 4 (aprile) e l'argomento giorno è passato come -1, l'interprete avrebbe usato 31 come giorno perché l'ultimo giorno del mese di marzo è 31.

Se l'argomento mese è passato come 3 (marzo) e l'argomento giorno è passato come -1, l'interprete si riferirebbe all'argomento Anno per determinare se l'anno è bisestile o meno. Ciò consentirebbe all'interprete di utilizzare sia 28 o 29 per il valore del giorno. L'interprete utilizza questo algoritmo per qualsiasi valore giorno passato come terzo parametro quando il numero è inferiore a 1.

Se il Giorno argomento è passato con un valore superiore a 28, 29, 30, o 31, l'interprete usa questo stesso algoritmo in senso inverso per determinare il mese e il giorno.

Conversione di un valore in Data

Se si dispone di un valore come quello fornito come una stringa e si desidera convertirlo in una data, è possibile richiamare la funzione **CDate** la cui sintassi è:

Function CDate(Value As Object) As Date

Questa funzione accetta qualsiasi tipo di valore, ma il valore deve essere convertibile in una data valida. Se la funzione riesce nella conversione, produce un valore Date. Se la conversione non riesce, si verifica un errore.
Come abbiamo visto finora, una data è un valore composto da tre parti: l'anno, il mese e il giorno. L'ordine di questi componenti e il modo in cui sono messi insieme per costituire una data riconoscibile dipendono dalla lingua e vengono definiti nel linguaggio e nelle Impostazioni internazionali nel Pannello di controllo.

L'Anno di una data

Il linguaggio Visual Basic supporta l'anno di una data che va da 1 a 9999. Ciò significa che questa è il Range che si può considerare quando si tratta di date nei fogli di lavoro. Nella maggior parte delle operazioni, durante la creazione di una data, se si specifica un valore compreso tra 1 e 99, l'interprete avrebbe usato il secolo in corso per la sinistra due cifre, ciò significa che, un anno come 4 o 04 indicherebbe l'anno 2004.

Nella maggior parte dei casi, per essere più precisi, di solito dovrete specificare sempre l'anno con 4 cifre. Se si dispone di un valore di data e si vuole scoprire l'anno, è possibile richiamare la funzione **Year**, la cui sintassi è:

Public Function Year(ByVal DateValue As Variant) As Integer

Come potete vedere, questa funzione assume un valore di data come argomento e dovrebbe contenere una data valida. In caso affermativo, la funzione restituisce l'anno in forma numerica di una data. Ecco un esempio:

```
Sub Prova()
  Dim data1 As Date
  data1 = #2/8/2004#
  MsgBox ("Sono nata nell'anno " & Year(data1))
End Sub
```

Ciò produrrebbe:

Fig. 8

Il mese di un anno

La parte del mese di una data è un valore numerico che va da 1 a 12 e quando si crea una data, è possibile specificarlo con 1 o 2 cifre. Se il mese è compreso tra 1 e 9, è possibile precederlo con uno 0. Se si dispone di un valore data e vuole ottenere il suo mese, è possibile chiamare la funzione **Month**, la cui sintassi è:

Function Month(ByVal DateValue As Variant) As Integer

Questa funzione richiede un oggetto Data come argomento e se la data è valida, la funzione restituisce un numero compreso tra 1 e 12 per il mese. Ecco un esempio:

```
Sub Prova()
    Dim data1 As Date
    data1 = #2/8/2004#
    MsgBox ("Il mese scelto è  " & Month(data1))
End Sub
```

Ciò produrrebbe:

Fig. 9

Come già menzionato, la funzione **Month** produce un valore numerico che rappresenta il mese di una data, se invece si vuole ottenere il nome del mese, è possibile chiamare la funzione **MonthName**, la cui sintassi è:

Function MonthName(ByVal Month As Integer, Optional ByVal Abbreviate As Boolean = False) As String

Questa funzione richiede due argomenti di cui uno opzionale. L'argomento richiesto deve rappresentare il valore di un mese e se è un valore valido, questa funzione restituisce il nome corrispondente. Ecco un esempio:

```
Sub Prova()
   Dim data1 As Date
   data1 = #2/8/2004#
   MsgBox ("Il mese scelto è  " & MonthName(Month(data1)))
End Sub
```

Ciò produrrebbe:

Fig. 10

Il secondo argomento consente di specificare se si desidera ottenere il nome breve o completo del mese. Il valore predefinito è il nome completo e in questo caso il valore predefinito dell'argomento è False, mentre se si desidera ottenere il nome breve, si deve passare il secondo argomento come True . Ecco un esempio:

```
Sub Prova()
Dim data1 As Date
   data1 = #2/8/2004#
   MsgBox ("Il mese scelto è  " & MonthName(Month(data1), True))
End Sub
```

Ciò produrrebbe:

Fig. 11

Il giorno di un mese

Il giorno è un valore numerico in un mese e a seconda del mese (e dell'anno), il suo valore può variare da 1 a 29 (febbraio bisestile), da 1 a 28 (Febbraio in un anno non bisestile), da 1 a 31 (gennaio, marzo, maggio, luglio, agosto, ottobre e dicembre), o da 1 a 30 (aprile, giugno, settembre e novembre).
Se si dispone di un valore di data e volete sapere il giorno in un anno, è possibile richiamare la funzione **Day**, la cui sintassi è:

Function Day(ByVal DateValue As Variant) As Integer

Questa funzione richiede una data come argomento e se la data è valida, la funzione restituisce il giorno numerico del mese dell'argomento data. Ecco un esempio:

```
Sub Prova()
   Dim data1 As Date
   data1 = #2/8/2004#
   MsgBox ("Il mese scelto è  " & Day(data1))
End Sub
```

Ciò produrrebbe:

Fig. 12

Il giorno della settimana

Per ottenere il nome del giorno della settimana, è possibile usare una funzione denominata **WeekdayName**, la cui sintassi è:

Function WeekdayName(ByVal Weekday As Integer, Optional ByVal Abbreviate As Boolean = False, Optional ByVal FirstDayOfWeekValue As Integer = 0) As String

Questa funzione richiede un argomento obbligatorio e due opzionali. L'argomento obbligatorio deve essere, o rappresentare un valore compreso tra 0 e 7. Se si passa come 0, l'interprete si riferisce alla lingua del sistema operativo per determinare il primo giorno della settimana, che in inglese americano è la Domenica. In caso contrario, la funzione riporterebbe il nome corrispondente del giorno. Ecco un esempio:

```
Sub Prova()
   MsgBox ("Il Giorno scelto è : " & WeekdayName(4))
End Sub
```

Ciò produrrebbe:

Fig. 13

Se si passa un valore negativo o un valore superiore a 7, si riceverà un errore. Il secondo argomento consente di specificare se si desidera ottenere il nome breve o completo. Il valore di default di questo argomento è **False**, che produce un nome completo, se si desidera il nome breve, si deve passare il secondo argomento come **True**. Ecco un esempio:

```
Sub Prova()
   MsgBox ("Il Giorno scelto è : " & WeekdayName(4, True))
End Sub
```

Per supportare più opzioni, il linguaggio Visual Basic fornisce la funzione **Format ()** la cui sintassi è:

Function Format(ByVal Expression As Object, Optional ByVal Style As String = "") As String

Il primo argomento è la data in cui deve essere formattata e il secondo argomento è una stringa che contiene la formattazione da applicare. Per crearla, si utilizza una combinazione del mese, giorno e anno. Ecco un esempio:

```
Sub Prova1()
   Dim data1 As Date
   data1 = #12/28/2014#
   MsgBox ("La data è  " & Format(data1, "dd MMMM, yyyy"))
End Sub
```

Ciò produrrebbe:

Fig. 14

Lavorare con stringhe di testo in Excel VBA

Un tipo di variabile che abbiamo toccato marginalmente in questo corso è il tipo **String** che come suggerisce il suo nome, viene utilizzato per contenere le stringhe di testo. Questo tipo di variabile viene utilizzata moltissimo, quindi vale la pena di approfondire questo argomento. Per impostare una variabile per contenere il testo è sufficiente usare la parola chiave **Dim** seguita dal nome della variabile e dalla notazione **As String** in questo modo: *Dim MyString As String*, mentre per memorizzare il testo all'interno della variabile è necessario circondarlo con virgolette doppie così: *MyString = "Testo"*

Si tenga presente che anche se si inseriscono dei numeri racchiusi dalle virgolette vengono trattati come testo e non come numeri: *MyString = "25"*, in questo modo viene assegnato il valore 25 come testo, e non memorizza nella variabile il numero 25. È inoltre possibile inserire il testo in una cella del foglio di calcolo in questo modo:

```
Dim MyString As String
MyString = "Testo"
ActiveCell.Value = MyString
```

Oppure si può ottenere il testo dalla cella del foglio di calcolo:

```
Dim MyString As String
MyString = ActiveCell.Value
```

Molto spesso, però, è necessario fare delle elaborazioni con il testo che si ottiene da una cella di un foglio di calcolo. Ad esempio, potrebbe essere necessario prendere un nome completo da una cella e posizionare il primo nome in un'altra cella e il cognome in un'altra. Per fare le cose in questo modo, è necessario sapere come utilizzare le funzioni stringa incorporate di Excel VBA.

Alcune funzioni elencate in questo articolo sono già state trattate in questo corso, vengono riprese in questo contesto per mostrarne l'uso con altre funzioni VBA

Le funzioni LCase e UCase

Le funzioni **Ucase** e **Lcase** vengono utilizzate per modificare le lettere in caratteri minuscoli o maiuscoli, dove *Ucase* converte tutti i caratteri in lettere maiuscole, mentre *Lcase* converte in lettere minuscole. Vediamo come funzionano con un esempio pratico, creiamo una nuova cartella di lavoro vuota e inseriamo alcune voci nelle celle A1, B1 e C1 come si vede in figura, e inseriamo un nome nella cella A2

	A	B	C
1	Testo	Lcase	Ucase
2	Nelson Mandela		
3			

Fig. 1

A questo punto entriamo nell'editor VBA e inseriamo una routine in un Modulo e scriviamo del codice per leggere il valore presente nella cella A2 come il seguente

```
Sub Prova1 ()
Dim Nome As String
Nome = Range ("A2"). Value
End Sub
```

Con questo codice abbiamo creato una variabile denominata *Nome*, che abbiamo dichiarato come String e per inserire un valore in questa variabile, abbiamo usato l'enunciato *Range ("A2"). Value*. Per convertire il testo in minuscolo, è necessario solo il comando: *LCase (Testo_da_convertire)*

Qualunque cosa si sta cercando di convertire va posto tra le parentesi tonde della funzione *LCase* e il testo che si sta tentando di convertire può essere immesso direttamente, circondato da virgolette, o in una variabile che contiene una stringa di testo. Se vogliamo inserire il testo presente nella cella A2 convertito nella cella B2, tutto quello che dobbiamo fare è scrivere un codice come il seguente:
Range ("A2"). Offset (, 1) .Value = LCase (Nome)

Con *Range ("A2"). Offset (, 1)* ci si sposta di una colonna a destra della cella A2, e poi accediamo alla proprietà *Value* della cella stessa e a destra del segno di uguale inseriamo la funzione LCase col nome della variabile che conterrà il testo da convertire e il codice sarà simile a questo

```
Sub Prova1 ()
Dim Nome As String
Nome = Range ("A2"). Value
Range("A2").Offset(, 1).Value = LCase(Nome)
End Sub
```

Se eseguiamo questa routine possiamo vedere nel foglio la conversione del testo che appare in questo modo

	A	B	C
1	Testo	Lcase	Ucase
2	Nelson Mandela	nelson mandela	
3			

Fig. 2

Al tempo stesso se vogliamo convertire il nome in maiuscolo il codice è molto simile, basta solo cambiare il nome della funzione in questo modo:*Range("A2").Offset(, 2).Value = UCase(Nome)*

Se guardiamo questa nuova riga di codice, notiamo che sono cambiate solo due cose, il valore di Offset, in quanto abbiamo un 2 invece di 1, questo perché ci si sposta di due colonne a destra della cella A2 e la funzione che converte in maiuscolo è *UCase*. Se si aggiunge questa la linea al codice e si manda in esecuzione la routine si ottiene

	A	B	C
1	Testo	Lcase	Ucase
2	Nelson Mandela	nelson mandela	NELSON MANDELA
3			

Fig. 3

Così ora abbiamo trasformato il valore della cella A2 a caratteri minuscoli e maiuscoli. Si noti che il nome in A2, Nelson Mandela, è scritto con la prima lettera maiuscola, in gergo si capitalizza la prima lettera di

ogni parola. Excel dispone della funzione di conversione, e usando VBA possiamo usare la funzione chiamata **Proper** che permette la capitalizzazione di una stringa utilizzando il seguente codice:

```
Dim Nome As String
Nome = "nelson mandela"
Range ("A2"). Offset (, 3) .Value = Application.WorksheetFunction.Proper (Nome)
```

	A	B	C	D
1	Testo	Lcase	Ucase	
2	Nelson Mandela	nelson mandela	NELSON MANDELA	Nelson Mandela
3				

Fig. 4

Il codice che converte la prima lettera in maiuscolo è: *Application.WorksheetFunction.Proper (Nome)*
Si consideri che Application è un oggetto di livello superiore, ovvero l'intero Excel,
mentre WorksheetFunction viene utilizzato per accedere alla funzione di Excel, mentre tra le parentesi tonde di Proper, si digita la variabile che si sta tentando di convertire. Così come la digitazione di un nome di variabile, è possibile digitare il testo direttamente circondato da virgolette.

Le funzioni Trim e Len

La funzione **Trim** viene utilizzata per tagliare lo spazio bianco indesiderato nel testo, quindi, se si ha la seguente stringa: " Un testo" usando Trim si eliminerebbero gli spazi e rimane "Un testo", mentre invece la funzione **Len** viene utilizzata per ottenere il numero di caratteri di una stringa. Se si inserisce il codice seguente in una routine:

```
Dim Nome As String
Dim LNome As Integer
Nome = "  Nelson Mandela  "
LNome = Len (Nome)
MsgBox LNome
```

Abbiamo creato due variabili, una chiamata *Nome* e una chiamata *LNome* e quest'ultima è stata dichiarata come Integer. Nella variabile Nome abbiamo archiviato il testo " Nelson Mandela ", ma abbiamo due spazi vuoti alla sinistra del nome e due spazi vuoti a destra, mentre la quarta linea è questa: *LNome = Len (Nome)*

In questa riga di codice stiamo utilizzando la funzione **Len** e tra le parentesi tonde abbiamo la variabile Nome. La funzione Len conterà quanti caratteri sono presenti nel testo che abbiamo memorizzato all'interno della variabile Nome. Quando VBA ha eseguito questo calcolo lo memorizza nella variabile denominata LNome. Poiché la funzione Len conta i caratteri, il valore restituito sarà un numero intero. Una volta eseguito il codice verrà mostrato in una finestra il numero 18, tuttavia, il nome Nelson Mandela è lungo 14 caratteri, la finestra di messaggio visualizza 18 perché ha contato gli spazi extra all'inizio e alla fine della stringa. Per rimuovere lo spazio, si deve utilizzare la funzione Trim in questo modo:*Nome = Trim (" Nelson Mandela ")*

Il testo o la variabile che si sta cercando di tagliare va tra le parentesi tonde alla destra della funzione e VBA quindi rimuovere qualsiasi spazio bianco dalla parte anteriore e dalla fine della stringa. Se si esegue di nuovo il codice la finestra di messaggio visualizza il valore di 14.

La funzione Space

A volte si rende necessario inserire uno spazio vuoto prima o dopo una stringa, per fare questo si usa la funzione space inserendo il numero di spazi tra le parentesi tonde. Ecco un codice per illustrare la funzione:

```
Dim Nome As String
Nome = "Nelson Mandela"
MsgBox Len(Nome)
Nome = Space(5) & Nome
MsgBox Len(Nome)
```

Eseguendo il codice viene visualizzata una finestra di messaggio che riporta il valore 14, che è il numero di caratteri che sono nel nome Nelson Mandela, mentre la seconda finestra di messaggio visualizza un valore di 19, che contiene i 14 caratteri originali, più i 5 aggiunti all'inizio del nome. Avremmo potuto aggiungere 5 spazi alla fine del nome in questo modo: *Nome = Nome & Space (5)*. Non deve trarre in inganno l'uso di due volte della variabile Name, infatti se analizziamo la riga di codice: *Nome & Space (5)*

Questa ci dice: "Prendi tutto ciò che è presente nella variabile denominata Nome e aggiungi 5 spazi."(Il simbolo & viene utilizzato per concatenare) e una volta che VBA ha concatenato il testo e lo spazio, si deve memorizzare da qualche parte, e il luogo in cui sarà memorizzato è alla sinistra del segno uguale. A sinistra del segno di uguale, abbiamo di nuovo la variabile Nome, pertanto qualunque sia stato il valore in precedenza della variabile sarà sostituito dal valore dalla destra del segno di uguale.

La Funzione Replace

Abbiamo già visto che la funzione **Replace** permette di sostituire il testo in una stringa con qualcos'altro. Supponiamo, per esempio, di avere una parola errata nella cella A5. È possibile utilizzare replace per modificare le lettere non corrette con quelle corrette.
È possibile utilizzare il foglio di lavoro per provarlo, aggiungendo due voci nelle celle A4 e B4 e digitare il titolo Originale nella cella A4 e la voce Sostituisci nella cella B4. Ora se inseriamo all'interno della cella A5 la parola errata Micrasaft, il foglio di calcolo dovrebbe apparire così:

	A	B	C
1	**Testo**	**Lcase**	**Ucase**
2	Nelson Mandela	nelson mandela	NELSON MANDELA
3			
4	**Originale**	**Sostituito**	
5	Micrasaft		
6			

Fig. 5

Per utilizzare la funzione Replace, sono necessari almeno tre argomenti tra la parentesi tonde:

Replace(string_to_search, string_to_replace, replace_with)

La prima cosa che serve è una stringa di testo da ricercare, poi si specifica il carattere o i caratteri che si vuole sostituire e infino il nuovo personaggio o personaggi. Con la funzione Replace si hanno anche tre optional che è possibile specificare. Queste sono: *start, count, compare.* I parametri opzionali vanno dopo il terzo elemento in sostituzione, con ciascuno dei quali è separato da una virgola:

Replace(string_to_search, string_to_replace, replace_with, start, count, compare)

Il parametro Start rappresenta il carattere della stringa in cui si desidera avviare la ricerca, il valore di default è il carattere 1, che è il primo carattere della stringa. Se si vuole iniziare da una posizione nella stringa diverso dal primo carattere, quindi è necessario digitare il numero di partenza.
Il parametro count è il numero di occorrenze da sostituire, il default è di sostituire ogni occorrenza presente in replace_with . Se si desidera solo per sostituire, le prime due ricorrenze quindi si digita il numero 2.

Il parametro compare ha tre opzioni: vbBinaryCompare, vbTextCompare, vbDatabaseCompare, ma non preoccupatevi di confrontare, in quanto viene utilizzato raramente. A titolo di esempio, aggiungiamo una nuova routine con il seguente codice

```
Dim OTesto As String
Dim CTesto As String
OTesto = Range("A5").Value
CTesto = Replace(OTesto, "a", "o")
Range("A5").Offset(, 1).Value = CTesto
```

Come si può vedere abbiamo due variabili stringa, Otesto e Ctesto, il valore per la variabile Otesto viene ricavato dalla cella A5 sul foglio di calcolo. Abbiamo poi abbiamo questa riga di codice: *CTesto = Replace (OTesto, "a", "o")*, in cui abbiamo la funzione Replace a destra del segno di uguale, per cui la prima voce tra le parentesi tonde del Replace è il nome della variabile Otesto e rappresenta il testo da ricercare. In seguito viene elencato il carattere che non è corretto (la lettera " a "), infine, abbiamo bisogno del nuovo testo che vogliamo nella stringa, che è la lettera " o ". Tutti e tre gli elementi sono separati da virgole. La riga finale inserisce il testo corretto nella cella B5 del foglio di calcolo. Eseguendo il codice il foglio di calcolo dovrebbe sembrare come questo:

	A	B	C
1	**Testo**	**Lcase**	**Ucase**
2	Nelson Mandela	nelson mandela	NELSON MANDELA
3			
4	**Originale**	**Sostituito**	
5	Micrasaft	Microsoft	
6			

Fig. 6

È possibile sostituire più di un carattere, se è necessario. Il seguente codice sostituisce l'errata Microsft con Microsoft: *Ctesto = Replace(Otesto, "sft", "soft")*

È possibile sostituire spazi nel testo digitando due virgolette. La prima serie di virgolette avrà uno spazio tra loro, mentre la seconda serie non ha spazio. Ad esempio: *Ctesto = Replace("M i c r o s o f t ", " ", "")*

Questa volta, la parola Microsoft ha uno spazio dopo ogni lettera, se vogliamo rimuovere gli spazi, il secondo parametro della funzione Replace è di due virgolette con uno spazio tra di loro, mentre il terzo parametro è di due virgolette senza spazio tra loro. Due doppi apici insieme significano "nessun carattere".

Le Funzioni InStr, InStrRev, StrReverse

InStr è l'abbreviazione di inString, e questa funzione stringa viene utilizzata per la ricerca di una stringa all'interno di un'altra. Necessita di almeno due elementi tra le parentesi tonde della funzione InStr: il testo da cercare, e ciò che si desidera trovare. VBA poi vi darà un intero in cambio.

Questo numero sarà 0 se la stringa non viene trovata, mentre se la stringa viene trovata, allora si ottiene la posizione di inizio della stringa di ricerca. Ecco un esempio per provare:

```
Dim Email As String
Dim Posizione As Integer
Email = "mia_email@prova.com"
Posizione = InStr(Email, "@")
MsgBox Posizione
```

Abbiamo dichiarato due variabili, una è una variabile Stringa che contiene un indirizzo email, e l'altra è un intero chiamato posizione. La linea di codice della funzione InStr linea è questa: *Posizione = InStr(Email, "@")*

La prima voce tra le parentesi tonde è la variabile email, mentre il secondo elemento è quello che vogliamo cercare, cioè la chiocciolina dell'indirizzo di posta elettronica. Se il simbolo @ non è nella variabile Email, VBA colloca uno 0 nella variabile Posizione. Quando si esegue il codice verrà visualizzato una finestra di messaggio con il numero 10, in quanto il simbolo @ è il decimo carattere della stringa indirizzo email. Ora se NON inseriamo il simbolo @ nella stringa della E-mail: *Email = "mia_emailprova.com"*

Eseguendo il codice la finestra di messaggio visualizza il valore 0, è possibile utilizzare questo risultato per un test di base sugli indirizzi di posta elettronica con questo codice:

```
Dim Email As String
Dim Posizione As Integer
Email = "mia_emailprova.com"
Posizione = InStr(Email, "@")

If Posizione = 0 Then
MsgBox "Non hai inserito un indirizzo email valido"
Else
MsgBox "Indirizzo email valido"
End If
```

Ci sono due parametri facoltativi per la funzione InStr e sono: *Start* e *Compare* e la sintassi è la seguente: *InStr(start, Text_To_Search, Find, comapre)*

Se si omette il numero di partenza, la funzione InStr cerca dall'inizio della stringa, mentre se si digita un numero per la partenza InStr inizia la ricerca da quel numero di carattere nella stringa.

Il parametro compare ha quattro opzioni:

- vbUseCompareOption
- vbBinaryCompare
- vbTextCompare
- vbDatabaseCompare

Questi parametri vengono utilizzati raramente. Simile a Instr c'è la funzione **InStrRev**, che sarebbe l'acronimo di Reverse, infatti questa funzione è la stessa di InStr con la sola differenza che InStrRev inizia la ricerca dalla fine della stringa invece che all'inizio.

La funzione StrReverse

Questa funzione è abbastanza facile da usare, come suggerisce il suo nome StrReverse inverte le lettere in una stringa di testo. Ecco un po' di codice per provare:

```
Dim Otesto As String
Dim Rtesto As String
Otesto = "Inserisci una frase"
Rtesto = StrReverse(Otesto)
MsgBox (Rtesto)
```

Quando viene eseguito il codice, la finestra di messaggio visualizzerà il testo invertito della variabile Otesto.

Le Funzioni Left e Right

Le funzioni **Left** e **Right** vengono utilizzate per tagliare i caratteri da una stringa. Si utilizzare Left per tagliare i caratteri dall'inizio della stringa, e Right per tagliare i caratteri a partire dalla fine della stringa. Tra le parentesi tonde di sinistra e destra della funzione si digita il numero di caratteri che si desidera tagliare. Qualche esempio può chiarire le cose.

```
Dim Email As String
Email = "mia_email@prova.com"
MsgBox Left(Email, 9)
MsgBox Right(Email, 9)
```

Nella prima riga si dichiara una variabile String e nella seconda si inserisce un indirizzo e-mail nella variabile Email, mentre nella terza linea si utilizza una finestra di messaggio che utilizza la funzione Left: *MsgBox Left(Email, 9)*

Quando si esegue il codice vedrete che la finestra di messaggio visualizza i primi 9 caratteri dell'indirizzo e-mail, partendo dalla sinistra del simbolo @. La quarta linea è questa: *MsgBox Right(Email, 9)*

La funzione di destra visualizza 9 caratteri a partire dal carattere finale dell'indirizzo e-mail. Questo è un esempio abbastanza semplice, adesso per un uso più complesso di Left e Right, supponiamo di avere un nome completo nella cella A1 in questo formato: Nelson Mandela, tuttavia, si supponga di voler avere il cognome prima del nome in questo modo: Mandela Nelson

È possibile utilizzare le funzioni Left, Right e InStr per raggiungere questo obiettivo. Creiamo una nuova routine e impostiamo quattro variabili, in questo modo:

```
Dim NomeC As String
Dim PNome As String
Dim SNome As String
Dim PosSpc As Integer
```

Posizionare il nome completo nella variabile NomeC: *NomeC = "Nelson Mandela"*. Ora usiamo la funzione InStr per individuare la posizione dello spazio nel nome: *PosSpc = InStr(NomeC, " ")*

Per ottenere solo il primo nome si può cominciare all'inizio del nome completo e andare fino alla posizione dello spazio (PosSpc) e togliere 1: *PNome = Left(NomeC, PosSpc - 1)*

Il motivo per cui è necessario detrarre 1 dalla variabile PosSpc è perché la funzione InStr restituisce la posizione dello spazio, un valore di 7 per il nostro nome, mentre invece il carattere finale del nome, è di 1 in meno di questo valore, infatti Nelson ha solo 6 caratteri.
Per ottenere il cognome, abbiamo bisogno di qualcosa di leggermente diverso. La posizione di partenza è la lunghezza del nome completo meno la lunghezza del nome, con questa operazione si otterrà il numero di caratteri partendo dalla destra del nome. Il codice è questo: *SNome = Right(NomeC, Len(NomeC) - Len(PNome))*. Così come il parametro finale di destra abbiamo questo codice: *Len(NomeC) - Len(PNome)*

Questo utilizza la funzione Len per ottenere la lunghezza delle variabili NomeC e PNome, infine, si visualizzano i risultati in una finestra di messaggio:*MsgBox (SNome & ", " & PNome)*

Ora abbiamo la variabile *SNome* e *PNome* separate dal simbolo di concatenazione (&), ma abbiamo anche bisogno di una virgola, e dobbiamo inserirla tra virgolette in modo che VBA lo vede come testo. Quindi stiamo dicendo, di inserire il cognome, poi una virgola, quindi il Nome. Inoltre si potrebbe anche aggiungere una funzione Trim per i nomi, per sbarazzarsi di qualsiasi spazio bianco. Come questo: *MsgBox (Trim(SNome) & ", " & Trim(PNome))*

Ma non è necessario per questo piccolo esempio, comunque si tenga presente anche questo aspetto nell'utilizzo futuro di queste funzioni. L'intero codice, quindi, dovrebbe essere simile a questo:

```
Sub Prova2()
Dim NomeC As String
Dim PNome As String
Dim SNome As String
Dim PosSpc As Integer

NomeC = "Nelson Mandela"
PosSpc = InStr(NomeC, " ")
```

Eseguendo il codice e si dovrebbe vedere questa finestra di messaggio:

Fig. 7

Il codice precedente funziona solo per i nomi che hanno due parti, se il nome è composto da 3 parti si deve usare un'altra funzione (Split)

La Funzione Mid

Questa funzione viene utilizzata per catturare i caratteri di una stringa di testo. Ha tre parti:

Mid(string_to_search, start_position, number_of_characters_to_grab)

La prima parte è la stringa che si desidera cercare e può essere un testo, una variabile oppure il testo diretto tra virgolette. La seconda parte è dove iniziare la ricerca della stringa e la parte finale è il numero di caratteri che si desidera catturare. Per dimostrare la funzione Mid, esaminare il seguente codice:

```
Dim Email As String
Dim Pcar As String
Email = "mia_emailprova.com"
Pcar = Mid(Email, 15, 4)
MsgBox Pcar
```

Abbiamo creato due variabili stringa, una chiamata Email e una chiamata Pcar, abbiamo poi memorizzato un indirizzo email nella variabile Email e poi viene il codice della funzione Mid: *Pcar = Mid(Email, 15, 4)*

Il testo che stiamo cercando è nella variabile Email variabile e vogliamo iniziare a catturare i caratteri dalla posizione 15 della stringa e Il numero di caratteri che vogliamo prendere è 4. Quando viene eseguito il programma, nella finestra di messaggio verrà visualizzato .com. La funzione Mid è molto utile in un Loop, in quanto permette di esaminare un carattere alla volta da una stringa di testo.

Esercizio con il Metodo String

Per ottenere una certa dimestichezza con i metodi String, vediamo un altro esercizio. Supponete di avere un codice prodotto su un foglio di calcolo che si presentava così: PD-23-23-45, tuttavia, è nel formato sbagliato. Per ottenere il codice giusto si devono rimuovere tutti i trattini, e poi si devono sostituire le lettere PD nel codice del prodotto con PDC, in modo che il codice si presenti simile a questo: PDC232345

La prima parte del problema è la rimozione dei trattini, abbastanza facile, basta usare Replace:

```
Dim Pcode As String
Pcode = "PD-23-2345"
Pcode = Replace(Pcode, "-", "")
```

Tra le parentesi tonde di Replace abbiamo il testo che vogliamo cercare, che è la variabile denominata Pcode, dopo abbiamo una virgola, e in seguito il carattere che vogliamo sostituire, il trattino, che viene sostituito con due doppie virgolette senza spazio tra loro.

La seconda parte del problema, si deve aggiungere la "C" dopo "PD", sarebbe facile, se VBA aveva una funzione di inserimento, ma non è così e questo rende il problema un po' più difficile. Ci sono alcuni modi per inserire la "C" nel posto giusto, noi lo faremo utilizzando le funzioni di stringa Left e Mid. Useremo Left per prendere i primi due caratteri, quindi aggiungeremo la "C", poi usiamo la funzione Mid per ottenere i numeri. Per ottenere i primi due caratteri e aggiungere la "C", il codice è questo:

```
Dim Ncar As String
Ncar = Left(Pcode, 2) & "C"
```

Partendo dalla sinistra della variabile Pcode, prendiamo due caratteri con il codice Left(Pcode, 2) e la lettera "C" viene aggiunta con la concatenazione in questo modo: Left(Pcode, 2) & "C". Il nuovo codice di tre lettere viene memorizzato nella variabile chiamata Ncar e per ottenere i numeri, utilizziamo Mid in questo modo:

```
Dim Num As String
Num = Mid(Pcode, 3)
```

La funzione Mid preleverà 3 caratteri dalla variabile Pcode e poiché non abbiamo specificato un numero finale, Mid prenderà il resto dei caratteri fino alla fine della stringa. L'unica cosa che resta da fare è quello di unire le due parti insieme:

```
Dim NewCar As String
NewCar = Ncar & Num
MsgBox NewCar
```

In questa parte si utilizza solo la concatenazione per unire la variabile Ncar e Num e l'ultima riga utilizza una finestra di messaggio per visualizzare i risultati. Tutto il codice, si presenta così:

```
Sub Prova3()
Dim Pcode As String
Dim Ncar As String
Dim Num As String
Dim NewCar As String
Pcode = "PD-23-2345"
Pcode = Replace(Pcode, "-", "")

Ncar = Left(Pcode, 2) & "C"
Num = Mid(Pcode, 3)
NewCar = Ncar & Num
MsgBox NewCar
End Sub
```

Quando si incontra un problema come quello sopra, la soluzione di solito è di utilizzare una delle funzioni Left, Right, o Mid (o tutti) di spezzare la stringa per poi unirli di nuovo insieme.

Le celle di un foglio di lavoro

Un foglio di calcolo è una serie di intersezioni verticali denominate colonne e orizzontali chiamate righe e un foglio è fatto di colonne e righe, che si intersecano, l'intersezione di una colonna e una riga crea un rettangolo chiamato **cella**:

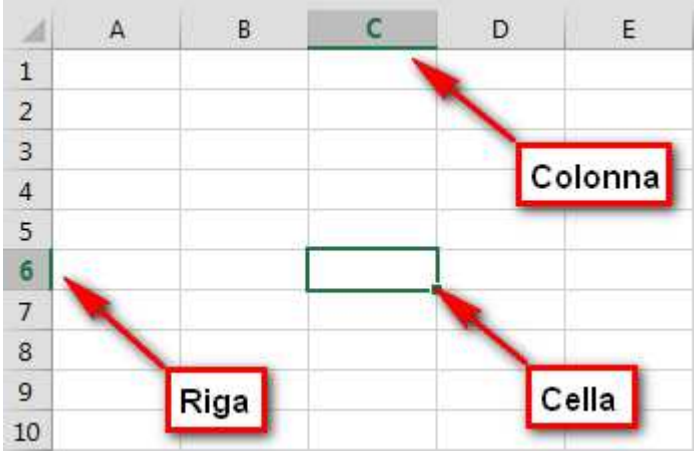


Fig. 1

Quando si avvia Excel, vengono create 16.384 colonne e 1.048.576 righe (dalla versione 2007 in poi) e come risultato, in un foglio elettronico avremmo a disposizione 17.179.869 (16,384 * 1.048.576) celle disponibili a cui è possibile accedervi utilizzando il mouse cliccando su una cella o con la tastiera premendo un tasto freccia. È inoltre disponibile un menu a più parti cliccando col tasto destro del mouse su una cella

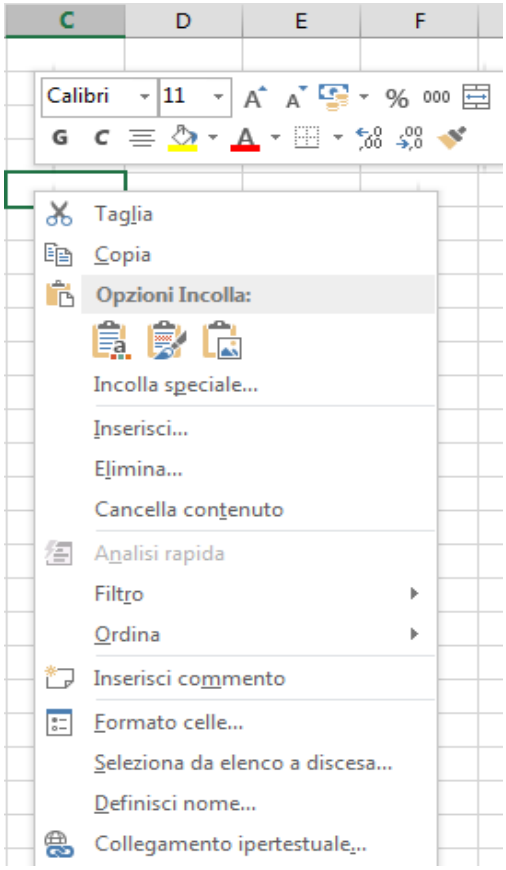


Fig. 2

Quando si utilizza un foglio di calcolo, per esempio quando si immettono le informazioni in esso, in realtà si sta lavorando su una cella, pertanto, in qualsiasi momento, si deve sempre essere in grado di identificare la cella che si sta utilizzando e per facilitare tale informazione, ogni cella ha un indirizzo, ottenuto dalla combinazione dell'etichetta della sua colonna e della sua riga, noto anche come nome primario della cella.

Posizione di una Cella

La combinazione del nome della colonna e l'etichetta della riga fornisce l'indirizzo o il nome di una cella e quando si clicca su una cella, la sua intestazione di colonna e di riga vengono evidenziate, inoltre per conoscere il nome (o indirizzo) di una cella, è possibile fare riferimento al Box, che si trova all'incrocio superiore sinistro tra colonne e righe

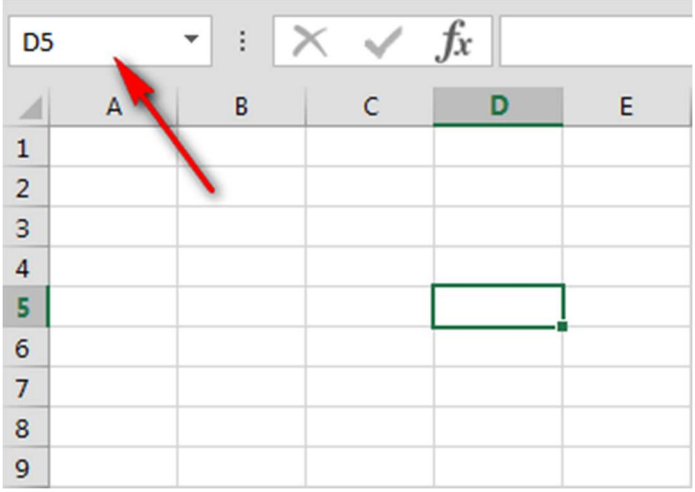


Fig. 3

Ogni volta che si lavora con le celle, e una di loro è selezionata i bordi appaiono più spessi rispetto alle altre celle, in questo modo la cella viene indicata come cella attiva. Per identificare visivamente la cella attiva, si può solo guardare la zona di lavoro e individuare la cella con bordi spessi. Nella schermata sopra, la cella attiva è D5 e In VBA, la cella attiva è rappresentato da un oggetto denominato **ActiveCell**.

Come già detto, prima di fare qualsiasi cosa su una cella, è necessario essere in grado di identificare o riconoscere la cellula, per fare questo visivamente, si può solo guardare, e individuare, la cella desiderata, ma per identificare a livello di programmazione una cella, si hanno diverse opzioni. È possibile identificare una cella utilizzando l'oggetto Range, a tale scopo, nelle parentesi dell'oggetto, si passa una stringa contenente il nome della cella. Ecco un esempio che fa riferimento alla cella situata come D6:

```
Sub Prova1()
   Workbooks.Item (1) .Worksheets.Item ("Foglio1"). Range ("D6")
End Sub
```

Per ottenere un riferimento a una cella, si dichiara una variabile di tipo *Range* e per inizializzare la variabile, si individua la cella e si assegna alla variabile utilizzando l'operatore ***Set***. Ecco un esempio:

```
Sub Prova1()
    Dim cella As Range
   Set cella = Workbooks.Item (1) .Worksheets.Item ("Foglio1"). Range ("D6")
End Sub
```

Le celle sono indicate come adiacenti quando si toccano e per fare riferimento a un gruppo di celle adiacenti, tra parentesi dell'oggetto Range, si passa una stringa che corrisponde all'indirizzo della cella che sarà sull'angolo, superiore sinistro seguito da due punti, seguito dall'indirizzo della cella posta nell'angolo inferiore destro. Ecco un esempio:

```
Sub Prova1()
    Range ("B2: H6")
End Sub
```

È possibile utilizzare questa stessa tecnica per fare riferimento a una cella, a tale scopo, si utilizza lo stesso indirizzo della cella da entrambi i lati dell'indirizzo. Ecco un esempio:

```
Sub Prova1 ()
    Range ("D4: D4")
End Sub
```

Invece di riferirsi ad un gruppo di celle adiacenti, è possibile fare riferimento a più di un gruppo di celle non adiacenti, Per fare questo, passare una stringa al Campo all'oggetto. Nella stringa, creare ogni intervallo come si vuole, ma separarli con virgole. Ecco un esempio:

```
Sub Prova1 ()
   Range ("D2: B5, F8: I14")
End Sub
```

Selezione di celle

Prima di fare qualsiasi cosa su una cella o un gruppo di celle, è necessario innanzitutto selezionarle, e la selezione delle celle è quasi equivalente a evidenziare una parola in un documento di testo. Vari mezzi sono disponibili per selezionare una cella o un gruppo di celle, è possibile utilizzare il mouse, la tastiera, o una combinazione di mouse e tastiera. Vediamo alcuni esempi:

- Per selezionare una cella usando il mouse, è sufficiente cliccare su di essa
- Per selezionare una cella utilizzando la tastiera, in quanto potrebbe essere necessario spostare lo stato attivo da una cella attiva a un'altra, premere i tasti freccia fino a selezionare la cella desiderata
- Per selezionare una cella in base al suo nome con il mouse e la tastiera, fare clic nella casella Nome e digitare il nome o l'indirizzo della cella
- Per selezionare la prima cella del documento utilizzando solo la tastiera, premere Ctrl + Home

Per supportare la selezione di celle, l'Intervallo oggetto è dotato di un metodo denominato **Select**, pertanto, per selezionare una cella in programmazione, dopo il riferimento cella, si richiama il metodo Select. Ecco un esempio:

```
Sub Prova1 ()
    Range ("D6"). Select
End Sub
```

Dopo aver selezionato una cella, viene memorizzato in un oggetto denominato ***Selection*** ed è possibile utilizzare questo oggetto per eseguire un'azione sulla cella che è attualmente selezionata. Invece di una sola cella, si consiglia di eseguire un'operazione comune in molte celle, il che significa che è necessario selezionarle prima.

È possibile selezionare le celle basate su colonne o su file ed è possibile selezionare le celle in una determinata regione; vale a dire, le celle adiacenti, oppure celle non adiacenti. Per selezionare tutte le celle di una colonna:

- Si deve cliccare sull'intestazione della colonna
- Oppure digitare il nome di una cella della colonna nella casella Nome e premere Invio, quindi premere il tasto Ctrl + barra spaziatrice
- Per selezionare tutte le celle di una serie di colonne adiacenti, si seleziona una colonna e si trascina il mouse nella direzione voluta tenendo premuto il tasto, una volta che la selezione è completa rilasciare il tasto del mouse

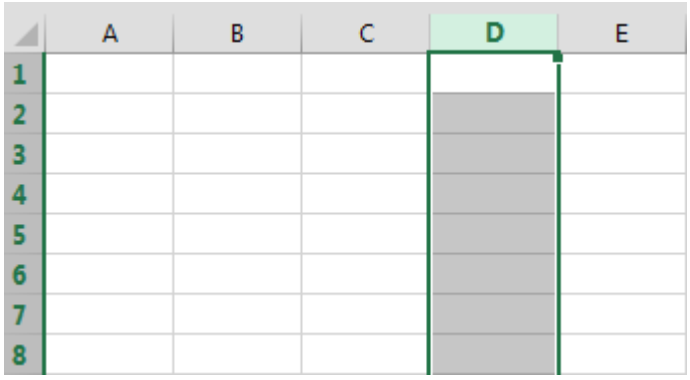


Fig. 4

Per selezionare tutte le celle di una riga:

- Si deve cliccare sull'intestazione di riga
- Oppure digitare il nome della cella di quella riga nella casella Nome e premere Invio e poi premere Shift + barra spaziatrice
- Per selezionare tutte le celle di una serie di righe, si seleziona la riga e si trascina il mouse nella direzione voluta tenendo premuto il tasto, una volta che la selezione è completa rilasciare il tasto del mouse

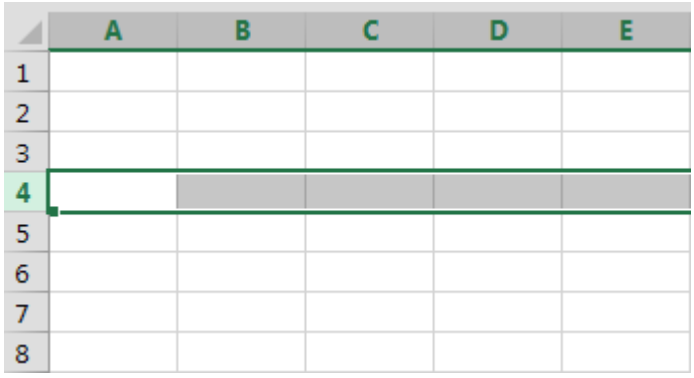


Fig. 5

Per selezionare le celle nella stessa regione

- Si deve cliccare su una cella e tenendo premuto il mouse sulla cella si trascina verso il basso o verso l'alto, a sinistra o a destra, fino all'ultima cella dell'intervallo
- Se si utilizza la tastiera, si deve premere i tasti freccia fino a selezionare la cella voluta che sarà in un angolo e tenendo premuto il tasto Shift, premere i tasti freccia sinistra, destra, in alto o in basso fino a raggiungere l'angolo opposto dell'intervallo e rilasciare il tasto Shift.

Per selezionare le celle non adiacenti, si deve cliccare su una delle celle e tenendo premuto il tasto Control si selezionano le celle desiderate, una volta che la selezione è completa, rilasciare Ctrl
Per selezionare tutte le celle di un foglio di lavoro, è possibile premere Ctrl + A, in alternativa, è possibile cliccare sul pulsante nel punto di intersezione della intestazione di riga e di colonna

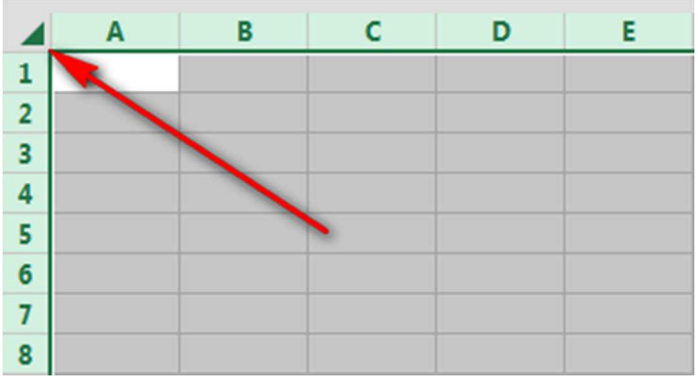


Fig. 6

Per selezionare a livello di codice un gruppo di celle adiacenti, si deve fare riferimento al gruppo con le tecniche che abbiamo visto in precedenza, quindi richiamare il metodo Select, mentre per selezionare a livello di codice tutte le celle di una colonna, si deve accedere alla colonna e usare il nome della colonna come stringa, quindi richiamare il metodo Select. Ecco un esempio:

```
Sub Prova1()
    ` selezionare tutte le celle dalla quarta colonna
    Columns (4) .Select
End Sub
```

Per eseguire questa operazione utilizzando il nome di una colonna(etichetta), si deve usare quel nome come argomento. Ecco un esempio che seleziona tutte le celle della colonna ADH:

```
Sub Prova1()
   `seleziona tutte le celle dalla colonna denominata ADH
    Columns ("ADH"). Select
End Sub
```

È inoltre possibile eseguire questa operazione utilizzando l'oggetto Range e per effettuare questa operazione, si deve utilizzare il Range immettendo il nome della colonna, seguito da due punti, seguito dal nome della colonna stessa. Ecco un esempio:

```
Sub Prova1()
    ` seleziona tutte le celle della colonna G
    Range ("G: G"). Select
End Sub
```

Per selezionare a livello di programmazione tutte le celle che appartengono a un gruppo di colonne adiacenti, si deve inserire tra le parentesi della colonne il nome della prima colonna da un lato, seguito da due punti ":" e poi il nome della colonna che sarà all'altra estremità. Ecco un esempio:

```
Sub Prova1()
   ` seleziona tutte le celle dell'intervallo dalla colonna D alla colonna G
   Columns ("D: G"). Select
End Sub
```

Per selezionare le celle che appartengono a un gruppo di colonne non adiacenti, utilizzano la tecnica che abbiamo visto in precedenza per fare riferimento alle colonne non adiacenti, quindi richiamare il metodo Select. Ecco un esempio:

```
Sub Prova1()
   ` seleziona le celle da colonne B, D e H
   Range ("H: H, D: D, B: B"). Select
End Sub
```

Per selezionare a livello di programmazione tutte le celle che appartengono a una riga, accedere a una riga, quindi richiamare il metodo Select. Ecco un esempio:

```
Sub Prova1()
   ` Seleziona tutte le celle della riga 6
    Rows (6) .Select
End Sub
```

È anche possibile utilizzare l'oggetto Range, dopo l'accesso alla riga, si richiama il metodo Select. Ecco un esempio che seleziona tutte le celle della riga 4:

```
Sub Prova1()
    Range ("4: 4"). Select
End Sub
```

Per selezionare tutte le celle che appartengono a un intervallo di righe, si deve fare riferimento all'oggetto Range e richiamare il metodo Select. Ecco un esempio che seleziona tutte le celle che appartengono alle righe da 2 a 6:

```
Sub Prova1()
    Rows ("2: 6"). Select
End Sub
```

Per selezionare tutte le celle che appartengono a righe non adiacenti, si deve fare riferimento alle righe e richiamare il metodo Select. Ecco un esempio che seleziona tutte le celle appartenenti alle righe 3, 5 e 8:

```
Sub Prova1()
    Range ("3: 3, 5: 5, 8: 8"). Select
End Sub
```

Per selezionare a livello di programmazione le celle nella stessa regione, si inserisce l'intervallo come stringa nell'oggetto Range, quindi si richiama il metodo Select. Ecco un esempio:

```
Sub Prova1 ()
    Range ("B2: H6"). Select
End Sub
```

Ricordate che è possibile utilizzare la stessa tecnica per fare riferimento a una sola cella e selezionarla. Ecco un esempio:

```
Sub Prova1()
   Range ("D4: D4").Select
End Sub
```

Per selezionare più di un gruppo di celle non adiacenti, si deve fare riferimento alla combinazione come abbiamo visto in precedenza e richiamare il metodo Select. Ecco un esempio:

```
Sub Prova1 ()
    Range ("D2: B5, F8: I14"). Select
End Sub
```

Per selezionare tutte le celle di un foglio di calcolo, è possibile richiamare il metodo Select sulle righe. Ecco un esempio:

```
Sub Prova1()
    Rows.Select
End Sub
```

Al posto delle righe, è possibile utilizzare le Colonne e si otterrebbe lo stesso risultato. Dopo aver selezionato un gruppo di celle, il gruppo viene memorizzato in un oggetto denominato Selection. È possibile utilizzare questo oggetto per intraprendere un'azione comune in tutte le celle che sono attualmente selezionate. Sappiamo che ognuno ha un nome fatto della combinazione del nome della colonna e il nome di una riga, se si desidera, è possibile modificare il nome di una cella, oppure è anche possibile creare un nome per un gruppo di celle.

Per conoscere o definire il nome di una cella, è possibile controllare la Casella Nome agendo in questo modo:

Cliccare sulla cella, e nella casella nome, inserire il nome e premere INVIO, Oppure sulla barra multifunzione, fare clic su Formule – Definisci Nomi, cliccare su Definisci nome e nella casella di testo Nome della finestra di dialogo Nuovo nome, digitare il nome desiderato e cliccare su OK

Oppure cliccare sulla cella nella cartella di lavoro e sulla barra multifunzione, cliccare su Formule – Definisci Nomi e nella finestra di dialogo Nuovo nome, nella casella di testo Nome, digitare il nome desiderato, nella casella combinata Ambito, accettare o specificare la cartella di lavoro, nella casella di testo Commento, digitare alcune parole se volete e nel campo Riferito a cliccare sul pulsante di destra, verrete riportati nella cartella di lavoro, selezionate la cella e cliccare poi di nuovo sul pulsante della finestra di dialogo e cliccare su OK

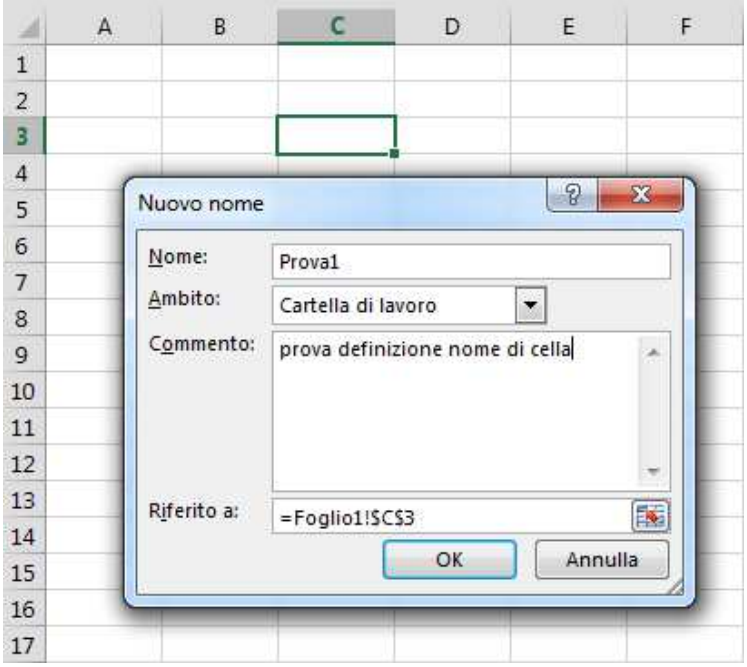


Fig. 7

Oppure sulla barra multifunzione, cliccare su **Formule – Gestione Nomi** e nella finestra di dialogo Gestione nomi, cliccare su Nuovo e si ripete quanto sopra specificato

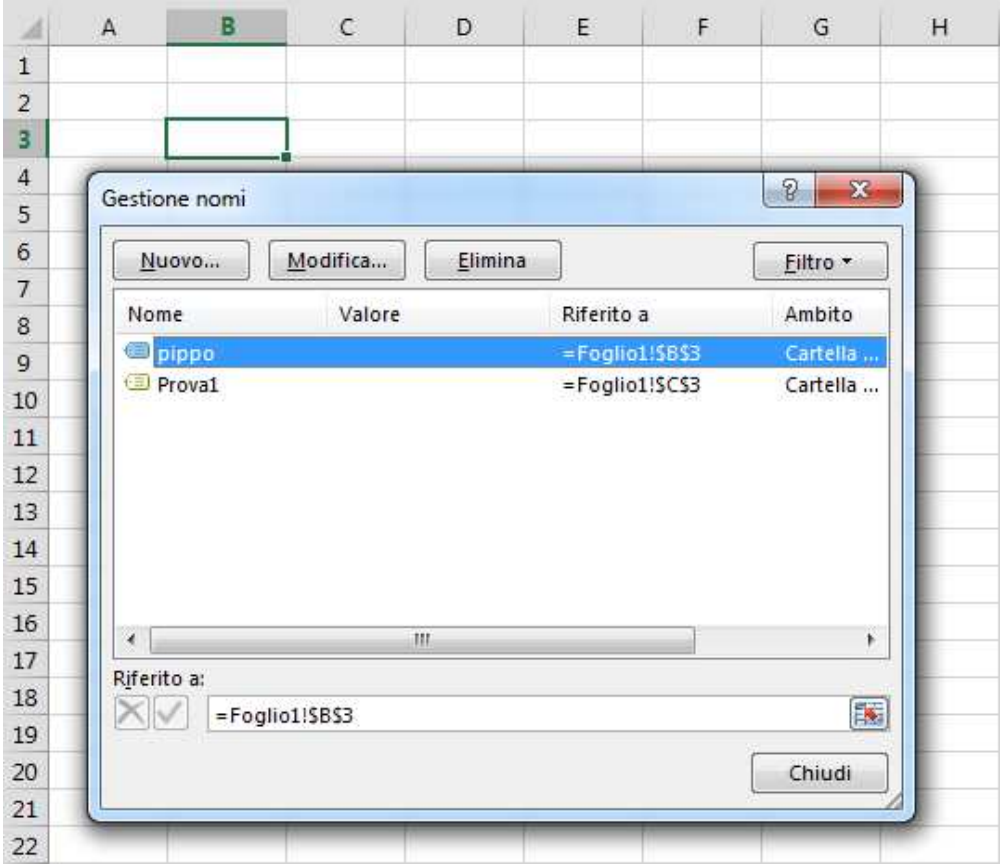


Fig. 8

Abbiamo già visto che, per fare riferimento a una cella utilizzando il suo nome, è possibile passare quel nome come stringa al Campo all'oggetto. Ora sappiamo come selezionare un gruppo di celle e se si seleziona più di una cella, il nome della prima cella viene visualizzato nella casella Nome. Nella maggior parte delle operazioni, questo non può essere utile, soprattutto se si desidera eseguire la stessa operazione su tutte le celle della selezione. Fortunatamente, Microsoft Excel consente di specificare un nome comune per il gruppo di celle selezionate. Per specificare un nome per un gruppo di celle:

Selezionare le celle con le tecniche che abbiamo visto in precedenza e nella casella nome, sostituire la stringa con il nuovo nome, Oppure selezionare le celle e dalla barra multifunzione, cliccare su Formule Definisci Nomi e nella casella di testo Nome della finestra di dialogo Nuovo nome, digitare il nome desiderato e cliccare su OK. Dopo aver creato un nome per un gruppo di celle, si può fare riferimento a quelle celle che utilizzano il nome, utilizzando l'oggetto Range e passare il nome come stringa. Ecco un esempio

```
Sub Prova1 ()
    Range ("pippo"). Select
End Sub
```

Le colonne di un foglio di lavoro

In VBA per Excel, per fare riferimento a livello di programmazione ad una colonna, si utilizza una **Collection** (raccolta) oppure un **Range** (Intervallo) e per ottenere un riferimento a una colonna o un gruppo di colonne, si dichiara una variabile di tipo Range

```
Sub Prova2()
    Dim Series As Range
End Sub
```

Per inizializzare la variabile, è possibile identificare le cartelle di lavoro e i fogli di lavoro che si stanno utilizzando, vediamo con degli esempi in cui utilizzeremo l'indice della colonna racchiuso tra le parentesi

```
Sub Prova2 ()
   ` si riferisce alla prima colonna
   Workbooks(1).Worksheets(2).Columns(1)
   ' si riferisce alla 12° Colonna
  Workbooks(1).Worksheets(2).Columns(12)
End Sub
```

È possibile omettere l'indice (1) per identificare la prima cartella di lavoro se sappiamo che si fa riferimento alla cartella di lavoro di default. Pertanto, il codice precedente può essere scritto come segue:

```
Sub Prova2 ()
   'Si riferisce alla quarta colonna
   Worksheets(2).Columns(4)
End Sub
```

Quando questo codice viene eseguito, Excel deve avere il secondo foglio di lavoro attivo, se così non fosse e il foglio attivo è un foglio di lavoro diverso dal secondo, si riceverà un errore:

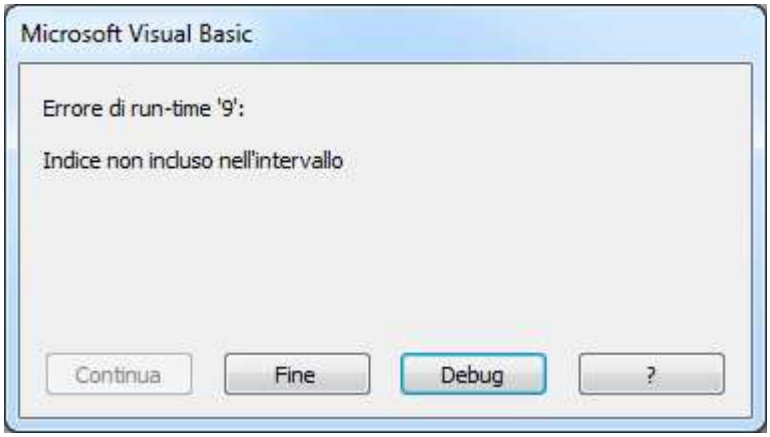


Fig. 1

Se si desidera accedere a una specifica colonna, in ogni foglio di lavoro della cartella attiva, è possibile omettere l'indicazione del foglio di lavoro. Ecco un esempio:

```
Sub Prova2()
   'Si riferisce alla quarta colonna
Columns(4)
End Sub
```

Questa volta, il codice indica che ci si riferisce alla quarta colonna di qualsiasi foglio di lavoro che è attualmente attivo. E' possibile fare riferimento a una colonna utilizzando il suo nome, passando la sua lettera o combinazione di lettere come una stringa tra le parentesi del metodo Columns in questo modo:

```
Sub Prova2()
   ` Questo si riferisce alla colonna A
Columns("A")
   'Questo si riferisce alla colonna denominata DR
   Columns("DR")
End Sub
```

Per fare riferimento a colonne adiacenti, è possibile utilizzare il metodo **Columns** e nelle parentesi si digita il nome della colonna all'estrema sinistra del Range, seguito da due punti "**:**" e seguito dal nome della colonna situata all'estrema destra del Range. Ecco un esempio che si riferisce alle colonne nell'intervallo da D a G:

```
Sub Prova2 ()
   ` riferito al Range delle colonne da D a G
   Columns("D:G")
End Sub
```

È inoltre possibile selezionare le colonne utilizzando il metodo **Range** e per effettuare questa operazione, si digita il nome della prima colonna, seguita da due punti, e seguito dal nome della colonna all'altra estremità

```
Sub Prova2 ()
   'si riferisce alle colonne nell'intervallo da D a H
   Range ("D: H")
End Sub
```

Per fare riferimento a colonne non adiacenti, si utilizza il metodo Range e nelle sue parentesi, si digita ogni nome di una colonna, seguito da due punti e seguito con lo stesso nome di colonna, quindi separare queste combinazioni con virgole in questo modo:

```
Sub Prova2()
  ` si riferisce alle colonne H, D, e B
   Range ("H: H, D: D, B: B")
End Sub
```

Selezionare colonne

Per supportare la selezione di una colonna, la classe Columns è dotata di un metodo denominato **Select** e non necessita di ricevere alcun argomento, semplicemente per selezionare la quarta colonna, si usa il codice come segue:

```
Sub Prova2()
   ` Seleziona la quarta colonna
   Columns(4). Select
End Sub
```

Mentre invece per selezionare una colonna utilizzando il suo nome, si usa questo codice:

```
Sub Prova2()
    ` seleziona la colonna ADH
    Columns("ADH").Select
End Sub
```

Quando viene selezionata una colonna, viene memorizzato in un oggetto chiamato **Selection**, ed è quindi possibile utilizzare l'oggetto per intraprendere l'azione da applicare alla colonna.
Per selezionare una serie di colonne adiacenti, nelle parentesi del metodo Columns, si immette il nome della prima colonna a un'estremità, seguito da due punti "**:**" e di seguito il nome della colonna che sarà all'altra estremità. Ecco un esempio:

```
Sub Prova2()
    'seleziona l'intervallo di colonne dalla Colonna D alla colonna G
    Columns("D: G"). Select
End Sub
```

È possibile utilizzare questa stessa notazione per selezionare una colonna utilizzando il metodo Range, inserendo nelle parentesi il nome della colonna, seguita da due punti, e seguito dal nome della stessa colonna. Ecco un esempio:

```
Sub Prova2 ()
    ` seleziona la Colonna G
    Range ("G: G"). Select
End Sub
```

Per selezionare invece delle colonne non adiacenti, si utilizza la tecnica che abbiamo visto in precedenza per fare riferimento alle colonne, quindi si richiama il metodo Select. Ecco un esempio:

```
Sub Prova2()
    ` seleziona le colonne B, D e H
    Range ("H: H, D: D, B: B"). Select
End Sub
```

Inserire colonne

Per sostenere la creazione di colonne, la classe Columns è dotata di un metodo denominato **Insert** e anche questo metodo non richiede alcun argomento o parametro, quando si richiama, è necessario specificare la colonna che succederà a quella nuova, per esempio se si vuole creare una nuova colonna nella terza posizione e spostare le colonne a destra si usa questo codice:

```
Sub Prova2 ()
     Columns(3). Insert
End Sub
```

Per aggiungere nuove colonne, si deve specificare la loro posizione utilizzando la classe Range come abbiamo visto prima, quindi richiamare il metodo Insert della classe Columns. Ecco un esempio che crea nuove colonne nelle posizioni delle colonne B, D e H che vengono spostate verso destra per far posto a quelle nuove

```
Sub Prova2 ()
     Range ("H: H, D: D, B: B"). Insert
End Sub
```

Eliminare colonne

Per fornire la capacità di eliminare una colonna, la colonna classe è dotata di un metodo denominato **Delete**. Per eliminare una colonna, si utilizza la raccolta Columns per specificare l'indice o il nome della colonna che verrà cancellata e richiamato il metodo Delete. Ecco un esempio che rimuove la quarta colonna.

```
Sub Prova2 ()
    Columns("D: D"). Delete
End Sub
```

Per eliminare varie colonne adiacenti, si deve specificare il Range delle colonne raccolta e richiamare il metodo Delete. Ecco un esempio:

```
Sub Prova2 ()
    Columns ("D: F"). Delete
End Sub
```

Per eliminare più colonne non adiacenti, si utilizza la classe Range richiamando il metodo Delete. Ecco un esempio che elimina le colonne C, E, e P:

```
Sub Prova2 ()
    Range ("C: C, E: E, P: P"). Delete
End Sub
```

Dimensioni delle colonne

Per supportare le dimensioni delle colonne, la classe Columns è dotata di una proprietà denominata **ColumnWidth**, pertanto si può modificare la larghezza di una colonna accedendo alla sua proprietà ColumnWidth e assegnare il valore desiderato. Ecco un esempio che imposta la larghezza della colonna di C a 4,50:

```
Sub Prova2 ()
    Columns ("C"). ColumnWidth = 4.5
End Sub
```

Per utilizzare la proprietà **Adatta** si deve prima selezionare la colonna e usare la selezione oggetto, accedere alla sua proprietà, quindi richiamare il metodo Adatta della proprietà della Colonna. Questo può essere fatto come segue:

```
Private Sub Prova2 ()
    Selection.Columns.AutoFit
End Sub
```

Per specificare la larghezza di molte colonne, si deve utilizzare la classe Range, quindi accedere alla proprietà **ColumnWidth**, e assegnare il valore desiderato. Ecco un esempio che imposta la larghezza delle colonne C, E e H a 5 per ciascuna colonna:

```
Sub Prova2 ()
    Range ("C: C, E: E, H: H"). ColumnWidth = 5 #
End Sub
```

Nascondere le colonne

Per nascondere una colonna si deve prima selezionarla, quindi assegnare il valore TRUE alla proprietà **Hidden** dell'oggetto EntireColumn della selezione. Si consideri il seguente codice:

```
Private Sub Prova2()
  'Questo codice nascondere la colonna F
   Columns ("F: F"). Select
   Selection.EntireColumn.Hidden = True
End Sub
```

Per nascondere una serie di colonne, si devono selezionare come abbiamo già visto, quindi assegnare il valore True (vero) alla proprietà **Hidden**. Si tenga presente che per rivelare le colonne nascoste si può agire anche cliccando col pulsante destro del mouse su un'intestazione di colonna e scegliere Scopri dal menu a discesa che compare, oppure dalla barra multifunzione, cliccando su Home e nella sezione celle, cliccare su Formato, posizionare il mouse su Nascondi & Scopri, quindi fare clic su Scopri colonne. Per visualizzare una colonna nascosta, si deve assegnare il valore False alla proprietà Hidden

```
Private Sub Prova2 ()
   Colonne ("F: F"). Select
   Selection.EntireColumn.Hidden = False
End Sub
```

Le Funzioni Condizionali

Una valida alternativa ad una condizione Vera o Falsa può essere la dichiarazione If ... Then ... ElseIf in quanto si comporta come l'espressione If ... Then ... Else, salvo che offre un maggior numero di scelte, se necessario. La sintassi è la seguente:

```
If Condizione1 Then
    Istruzioni1
ElseIf Condizione2 Then
   Istruzioni2
ElseIf CondizioneX Then
   IstruzioniX
End If
```

Quando viene utilizzata questa istruzione, verrà esaminata innanzitutto *Condizione1*, se risulta essere Vera, il programma eseguirà *Istruzioni1* e cessa l'esame delle altre condizioni, mentre se Condizione1 è falsa, verrà esaminata Condizione2 e nel caso risulti essere vera verrà eseguita Istruzioni2. In sostanza, ogni volta che una condizione è falsa, il programma continuerà l'esame delle condizioni fino a trovarne una che sia vera e una volta che è stata trovata verrà eseguita la sua istruzione e il programma terminerà l'esame condizionale quando incontra l'istruzione End If . Ecco un esempio:

```
Sub Test()
    Dim eta As Byte
    eta = 32
    If eta <= 18 Then
        MsgBox ("Fascia Età : " & "Ragazzo")
    ElseIf eta < 55 Then
        MsgBox ("Fascia Età : " & "Adulto")
    End If
End Sub
```

Ciò produrrebbe:

Fig. 1

Esiste però la possibilità che nessuna delle condizioni indicate sia vera, in questo caso, è necessario fornire una condizione "generica" e viene fatta con un ultimo Else, che deve essere l'ultima nella lista di condizioni e deve agire se nessuna delle condizioni precedenti è risultata vera. La formula da usare potrebbe essere:

```
If Condizione1 Then
    Istruzioni1
ElseIf Condizione2 Then
   Istruzioni2
ElseIf CondizioneX Then
   IstruzioniX
Else
    CondizioneG
End If
```

Ecco un esempio:

```
Sub Test()
    Dim eta As Byte
    eta = 65
    If eta <= 18 Then
        MsgBox ("Fascia Età : " & "Ragazzo")
    ElseIf eta < 55 Then
        MsgBox ("Fascia Età : " & "Adulto")
    Else
        MsgBox ("Fascia Età : " & "Anziano")
    End If
End Sub
```

Ciò produrrebbe:

Fig. 2

Dichiarazioni condizionali e funzioni

Come abbiamo già visto, sappiamo che una funzione viene utilizzata per eseguire un compito specifico e produrre un risultato. Ecco un esempio:

```
Private Function etaF()
    Dim eta
    eta = InputBox("Inserisci l'età")
    etaF = ""
End Function
```

Quando una funzione esegue il suo compito, può incontrare diverse situazioni, alcune delle quali avrebbero bisogno di essere controllate per la veridicità o la negazione, ciò significa che le istruzioni condizionali possono aiutare una procedura con la sua assegnazione. Per meglio comprendere, sappiamo che una funzione ha lo scopo di restituire un valore, a volte però deve svolgere alcuni compiti i cui risultati potrebbero portare a risultati diversi.

Una funzione può restituire un solo valore (abbiamo visto che, passando argomenti per riferimento, è possibile effettuare una procedura di restituire più di un valore), ma si può fare il rendering di un risultato a seconda di un particolare comportamento. Se una funzione richiede una risposta da parte dell'utente, in quanto l'utente può fornire risposte diverse, è possibile trattare ogni risultato diverso. Si consideri la seguente funzione:

```
Private Function etaF()
    Dim eta
    eta = InputBox("Inserisci l'età")
    If eta <= 18 Then
        etaF = "Ragazzo"
    ElseIf eta < 55 Then
        etaF = "Adulto"
    End If
End Function

Sub Test()
    Dim tipo
    tipo = etaF
    MsgBox ("Fascia di età : " & tipo)
End Sub
```

A prima vista, questa funzione sembra a posto. L'utente è invitato a fornire un numero. Se l'utente inserisce un numero inferiore a 18 (escluso), la funzione restituisce "Ragazzo". Ecco un esempio del programma:

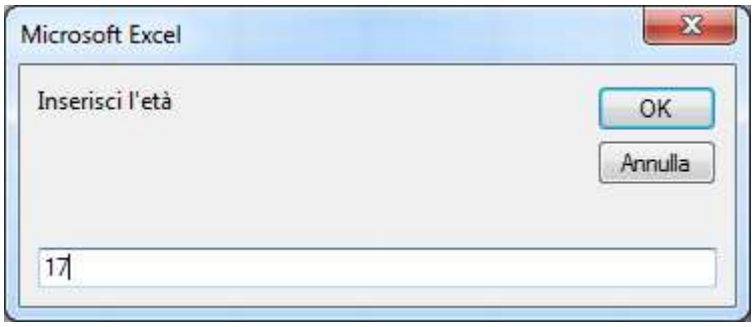


Fig. 3

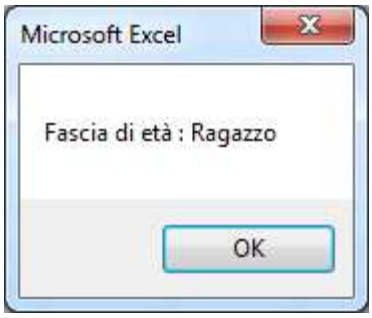


Fig. 4

Se invece l'utente fornisce un numero compreso tra 18 (incluso) e 55, la funzione restituisce "Adulto". Ecco un altro esempio del programma:

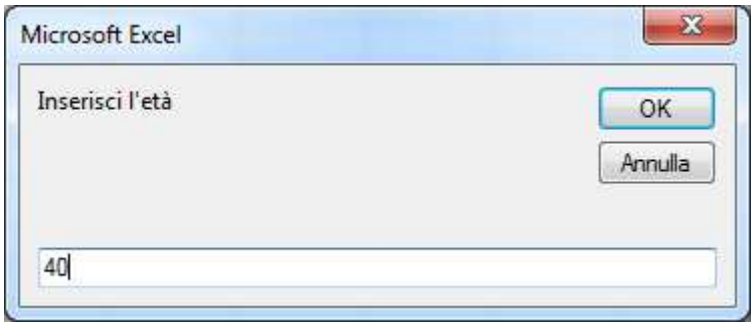


Fig. 5

Fig. 6

Si deve tener presente che i valori restituiti sono dovuti esclusivamente alle istruzioni condizionali inserire e non alla funzione, e che la condizione viene verificata quando è Vera, pertanto se viene inserito un valore che non si "adatta" alle istruzioni la funzione non restituisce nessun valore.

Nel nostro esempio, se l'utente inserisce un numero superiore a 55 (escluso), la funzione non eseguirà nessuna delle dichiarazioni, ciò significa che l'esecuzione raggiungerà la linea End Function senza trovare un valore da restituire. Ecco un altro esempio del programma:

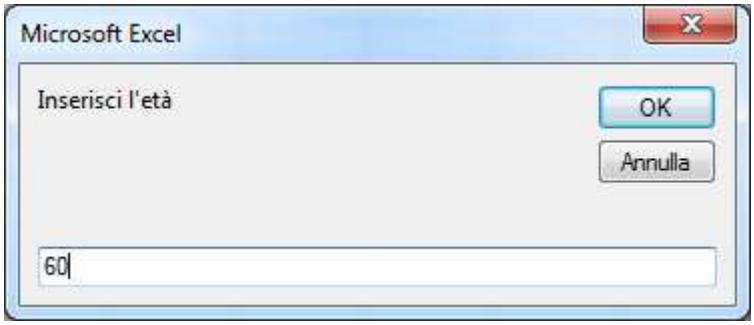


Fig. 7

Fig. 8

Per risolvere questo problema, si dispone di varie alternative, se la funzione utilizza una condizione If ... Then, è possibile creare una condizione Else che abbraccia un valore diverso da quelli validati in precedenza. Ecco un esempio:

```
Private Function etaF()
    Dim eta
    eta = InputBox("Inserisci l'età")
    If eta <= 18 Then
        etaF = "Ragazzo"
    ElseIf eta < 55 Then
        etaF = "Adulto"
    Else
        etaF = "Anziano"
    End If
End Function

Sub Test()
    Dim tipo
    tipo = etaF
    MsgBox ("Fascia di età : " & tipo)
End Sub
```

Questa volta, la condizione Else viene eseguita se nessun valore si può applicare alle condizioni If o ElseIf. Ecco un altro esempio:

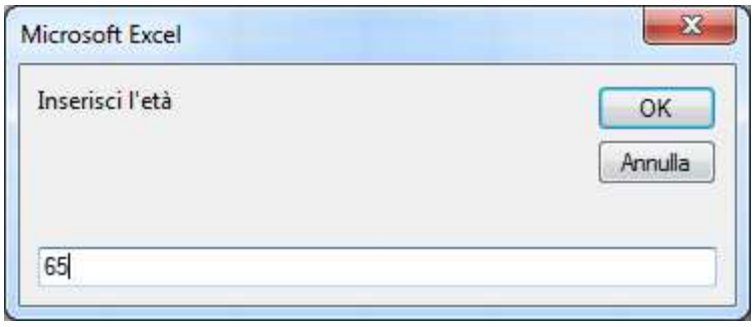


Fig. 9

Fig. 10

Un'altra alternativa è quella di fornire un ultimo valore di ritorno prima della linea End Function, in questo caso, se l'esecuzione raggiunge la fine della funzione, troverebbe dei dati da restituire, ma si dovrebbe conoscere il dato da restituire.

Ciò dovrebbe essere fatto nel modo seguente:

```
Private Function etaF()
    Dim eta
    eta = InputBox("Inserisci l'età")
    If eta <= 18 Then
        etaF = "Ragazzo"
    ElseIf eta < 55 Then
        etaF = "Adulto"
     End If

     etaF = "Anziano"
End Function
```

Se la funzione utilizza una condizione If, entrambe le implementazioni produrrebbero lo stesso risultato.

Uso della funzione Immediata IIF

La funzione **IIf** può essere usata anche al posto di uno scenario If ... Then ... ElseIf, infatti quando la funzione viene richiamata, l'espressione viene controllata e, come abbiamo già visto, se l'espressione è vera, la funzione restituisce l'argomento True e ignora l'ultimo argomento. Per utilizzare questa funzione come alternativa alla dichiarazione If ... Then ... ElseIf, se l'espressione è falsa, anziché immediatamente restituire il valore dell'argomento False, è possibile tradurre la parte in una nuova funzione IIf. La sintassi diventerebbe:

```
Public Function IIf( _
   ByVal Expression As Boolean, _
   ByVal TruePart As Object, _
         Public Function IIf( _
            ByVal Expression As Boolean, _
            ByVal TruePart As Object, _
            ByVal FalsePart As Object _
         ) As Object
) As Object
```

In questo caso, se l'espressione è falsa, la funzione restituisce il valore True e si ferma. Ecco esempio:

```
Sub Test()
    Dim eta As Byte, fasciaE As String
    eta = 74

    fasciaE = IIf(eta <= 18, "Ragazzo", IIf(eta < 55, "Adulto", "Anziano"))
    MsgBox ("Fascia eta : " & fasciaE)
End Sub
```

Abbiamo visto che in un If ... Then ... ElseIf è possibile aggiungere il numero di condizioni ElseIf come si desidera, nello stesso tempo, è possibile richiamare il numero di funzioni IIF che si ritengono necessarie:

```
Public Function IIf( _
   ByVal Expression As Boolean, _
   ByVal [COLOR="rgb(139, 0, 0)"]TruePart [/COLOR]As Object, _
         Public Function IIf( _
            ByVal Expression As Boolean, _
            ByVal [COLOR="rgb(139, 0, 0)"]TruePart [/COLOR]As Object, _
                 [COLOR="rgb(139, 0, 0)"]Public Function IIf[/COLOR]( _
                    ByVal Expression As Boolean, _
                    ByVal [COLOR="rgb(139, 0, 0)"]TruePart [/COLOR]As Object, _
                         [COLOR="rgb(139, 0, 0)"]Public Function IIf[/COLOR]( _
                            ByVal Expression As Boolean, _
                            ByVal TruePart As Object, _
                            ByVal [COLOR="rgb(139, 0, 0)"]FalsePart [/COLOR]As Object _
                         ) As Object
                 ) As Object
         ) As Object
) As Object
```

Scegliere un valore alternativo

Come abbiamo visto finora, la funzione Choose prende una lista di argomenti e per usarla come alternativa al If ... Then ... ElseIf ... ElseIf, è possibile passare i valori che si giudicano necessari per il secondo argomento. L'indice del primo elemento del secondo argomento sarebbe 1, mentre l'indice del secondo elemento del secondo argomento sarebbe 2, e così via. Quando la funzione viene richiamata, si dovrebbe prima ottenere il valore del primo argomento, allora si devono controllare gli indici dei membri disponibili del secondo argomento. Ecco un esempio:

```
Sub Test()
    Dim statoC As Byte, tipo As String
    statoC = 3
    tipo = Choose(statoC, "Tempo Pieno", "Part Time", "Stagionale", "A chiamata")

    MsgBox ("Tipo di Inquadramento : " & tipo)
End Sub
```

Ciò produrrebbe:

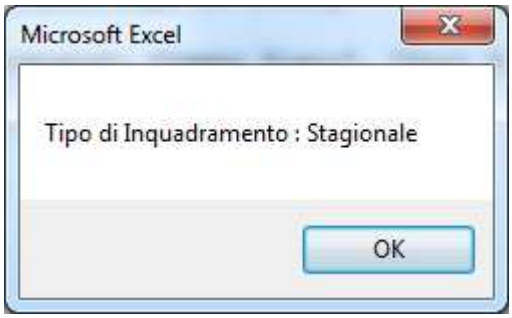


Fig. 11

Finora, abbiamo usato solo le stringhe per i valori del secondo argomento della funzione **Choose**, in realtà, i valori del secondo argomento può essere anche nullo, può essere una costante o una stringa.

Ecco un esempio:

```
Private Function Moperat()
   Moperat = "**-- Lista Operatori --**" & vbCrLf & vbCrLf & _
                 "James Bond" & vbCrLf & _
                 "Eva Kant" & vbCrLf & _
                 "Arsenio Lupin" & vbCrLf & _
                 "Rocky Balboa" & vbCrLf & _
                 "Sandro Pertini"
End Function

Sub Test()
   Dim tipo As Byte, Result$
   tipo = 3
   Result = Choose(tipo, _
               "Tipo Contratto : Full Time", _
               "Tipo Contratto : Part Time", _
               Moperat, _
               "Tipo Contratto : Stagionale")
   MsgBox (Result)
End Sub
```

Ciò produrrebbe:

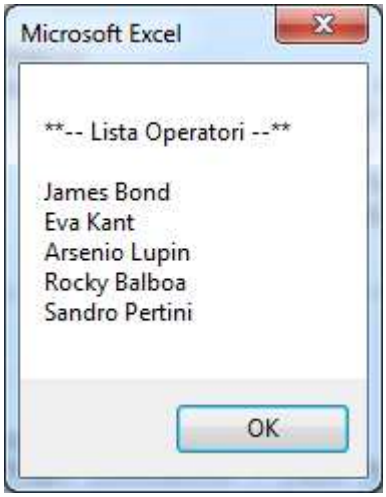


Fig. 12

I valori del secondo argomento possono anche essere di diversi tipi.

Il passaggio a un valore alternativo

La funzione **Switch** è un metodo alternativo alla condizione If ... Then ... ElseIf ... ElseIf e l'argomento di questa funzione viene passato come un elenco di valori, come visto in precedenza, ogni valore viene passato come una combinazione di due valori:

CondizioneX, StatementX

Come si accede alla funzione, il compilatore controlla ogni condizione, se una condizione è vera, la sua dichiarazione viene eseguita, e se una condizione Y è falsa, il compilatore la salta. È anche possibile fornire un maggior numero di queste combinazioni.

Ecco un esempio:

```
Private Enum tipoC
    TPn
    PTm
    Stg
    Ach
End Enum

Sub Test()
    Dim tipo As tipoC, Result As String

    tipo = tipoC.Stg
    Result = "Sconosciuto"

    Result = Switch(tipo = tipoC.TPn, "Tempo Pieno", _
                    tipo = tipoC.PTm, "Part Time", _
                    tipo = tipoC.Stg, "Stagionale", _
                    tipo = tipoC.Ach, "A Chiamata")

    MsgBox ("Tipo Contratto : " & Result)
End Sub
```

Ciò produrrebbe:

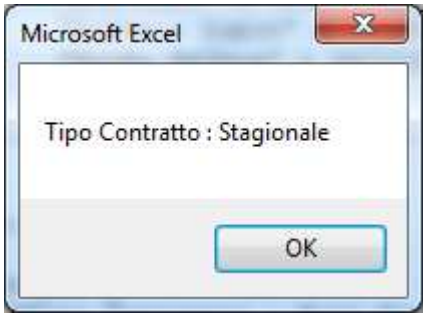


Fig. 13

In una condizione If ... Then ... ElseIf ... ElseIf, abbiamo visto che c'è una possibilità che nessuna delle condizioni si adatterebbe, nel qual caso è possibile aggiungere una ultima dichiarazione Else. La funzione Switch supporta anche questa situazione se si utilizza un numero, un carattere o una stringa e per fornire questa ultima alternativa, invece di un CondizioneX, si deve immettere un valore Vero e includere la dichiarazione. Ecco un esempio:

```
Sub Test()
   Dim tipo As Byte, Result As String
   tipo = 12
   Result = Switch(tipo = 1, "Tempo Pieno", _
                   tipo = 2, "Part Time", _
                   tipo = 3, "A Chiamata", _
                   tipo = 4, "Stagionale", _
                   True, "Sconosciuto")

   MsgBox ("Tipo Contratto : " & Result)
End Sub
```

Ciò produrrebbe:

Fig. 14

Ricorda che puoi utilizzare anche un carattere. Ecco un esempio:

```
Sub Test()
   Dim sesso As String, Result As String
   sesso = "H"
   Result = Switch(sesso = "f", "Femmina", _
                   sesso = "F", "Femmina", _
                   sesso = "m", "Maschio", _
                   sesso = "M", "Maschio", _
                   True, "Sconosciuto")

   MsgBox ("Tipo Sesso: " & Result)
End Sub
```

Ciò produrrebbe:

Fig. 15

Le Funzioni Left, Right, Mid, Len e Replace

In VBA una stringa si riferisce ad una sequenza di caratteri adiacenti all'interno di virgolette vale a dire, "*Questa è un'espressione stringa racchiusa tra virgolette, in VBA*." Questi caratteri sono letteralmente interpretati come caratteri, nel senso che questi rappresentano i personaggi stessi piuttosto che i loro valori numerici. Una stringa può includere lettere, numeri, spazi e punteggiatura e un'espressione stringa può avere come suoi elementi - una stringa di caratteri adiacenti, una funzione che restituisce una stringa, una variabile stringa, una stringa costante o una variante stringa.

La Funzione LEFT

La funzione **Left** in Excel può essere utilizzata sia come funzione di foglio che come una funzione VBA e restituisce il numero specificato di caratteri di una stringa di testo, a partire dal primo carattere più a sinistra. Si può utilizzare questa funzione per estrarre una sotto-stringa iniziando dalla parte sinistra di una stringa di testo usando la seguente sintassi:

LEFT (text_string, char_numbers)

È necessario menzionare l'argomento *text_string* che rappresenta la stringa di testo da cui si desidera estrarre il numero specificato di caratteri. L'argomento *char_numbers* è facoltativo quando si usa come una funzione del foglio di lavoro, che specifica il numero di caratteri da estrarre dalla stringa di testo, ricorda che il valore char_numbers deve essere uguale o maggiore di zero, se è maggiore della lunghezza della stringa di testo, la funzione LEFT restituirà la stringa di testo integralmente e se viene omesso, riporterà il valore predefinito che è pari a 1. Se invece si utilizza come una funzione VBA, è necessario specificare gli argomenti, e se text_string contiene Null, (cioè stringa vuota) la funzione restituisce una stringa vuota.

La Funzione RIGHT

La funzione **Right** in Excel può essere utilizzata sia come funzione di foglio che come una funzione VBA e restituisce il numero specificato di caratteri di una stringa di testo, a partire dall'ultimo carattere più a destra. Si può utilizzare questa funzione per estrarre una sotto-stringa iniziando dalla parte destra di una stringa di testo usando la seguente sintassi:

RIGHT (text_string, char_numbers)

È necessario inserire l'argomento *text_string* che rappresenta la stringa di testo da cui si desidera estrarre il numero specificato di caratteri. L'argomento*char_numbers* è facoltativo quando si usa come una funzione del foglio di lavoro, che specifica il numero di caratteri da estrarre dalla stringa di testo, tenendo presente che il valore char_numbers deve essere uguale o maggiore di zero, se è maggiore della lunghezza della stringa di testo, la funzione RIGHT restituirà la stringa di testo integralmente, mentre se viene omesso, verrà utilizzato il valore predefinito che è pari a 1. Se invece si utilizza come una funzione VBA, è necessario specificare gli argomenti, e se text_string contiene Null (cioè una stringa vuota), la funzione restituisce una stringa vuota

La Funzione MID

La funzione **Mid** può essere utilizzata sia come funzione di foglio che come una funzione VBA e restituisce il numero specificato di caratteri di una stringa di testo, a partire da una posizione specificata, cioè a partire da un numero di caratteri specificato.

Si può utilizzare questa funzione per estrarre una sotto-stringa iniziando da qualsiasi parte di una stringa di testo usando la seguente Sintassi:

MID (text_string, start_number, char_numbers)

L'argomento *text_string* indica la stringa di testo da cui si desidera estrarre il numero specificato di caratteri, mentre l'argomento *start_number* specifica il numero di caratteri da cui iniziare l'estrazione della sotto stringa, dove il primo carattere della stringa di testo deve essere indicato dal valore di start_number e incrementando verso destra di 1, mentre l'argomento *char_numbers* specifica il numero di caratteri da estrarre dalla stringa di testo.

Se start_number è maggiore della lunghezza della stringa di testo, viene restituita una stringa vuota (lunghezza zero), mentre se è minore della lunghezza della stringa di testo ma con un valore di char_numbers maggiore della lunghezza della stringa di testo, la funzione MID restituirà la stringa di testo integralmente dalla posizione start_number fino alla fine della stringa di testo.

Quando la funzione MID viene usata come una funzione nel foglio di lavoro e se viene specificato un valore negativo per char_numbers, MID restituirà il valore di errore (numero di errore), come pure start_number, se viene posto inferiore a 1, MID restituirà il valore di errore. Quando si utilizza come una funzione del foglio di lavoro, tutti gli argomenti devono essere specificati. Quando invece la funzione MID viene usata come una funzione VBA, l'argomento char_numbers è facoltativo e se viene omesso la funzione restituisce la stringa di testo per intero dalla posizione start_number alla fine della stringa di testo e tutti gli altri argomenti devono essere specificati, inoltre se text_string contiene una stringa vuota (Null), la funzione restituisce una stringa vuota.

La Funzione LEN

La funzione **Len** può essere utilizzata sia come funzione di foglio che come una funzione VBA e restituisce il numero di caratteri di una stringa di testo. Si utilizzare questa funzione per ottenere la lunghezza di una stringa di testo con questa sintassi:

LEN (text_string)

ed è necessario menzionare l'argomento *text_string* che corrisponde alla stringa di testo di cui si desidera conoscerne la lunghezza espressa in numero di caratteri, si noti che vengono contati come caratteri anche gli spazi vuoti. Mentre usando LEN come una funzione VBA con la sintassi: *Len (text_string)* o *Len (variable_name)*, è possibile utilizzare una stringa di testo o un nome di una variabile e la funzione restituisce un valore Long che rappresenta il numero di caratteri contenuti nella stringa o il numero di byte necessari per memorizzare una variabile. Utilizzando la funzione Len per una variabile di tipo Variant, VBA tratterà la variabile come una stringa e restituisce il numero di caratteri contenuti nella stessa. Si tenga presente che se l'argomento text_string o la variabile utilizzata contiene una stringa vuota (Null), anche il valore di ritorno sarà una stringa vuota.

Esempio delle funzioni LEN e MID, per determinare i caratteri che compaiono in posizioni dispari in una stringa di testo

```
Sub prova3()
Dim str As String, i As Integer
str = ActiveSheet.Range("A2")
 For i = 1 To Len(str)
'Controllare le posizioni dispari
If i Mod 2 = 1 Then
'Carattere di ritorno alla posizione dispari
MsgBox Mid(str, i, 1)
End If
Next
End Sub
```

Esempio delle funzioni Left, Right, Mid e Len

```
Sub prova1()
Dim str As String, strL As String, strR As String, strM As String
str = "Bepi Rua"
strL = Left(str, 7)
'Restituisce "Bepi Ru", che sono i primi 7 caratteri
MsgBox strL
strL = Left(str, 15)
'Restituisce "Bepi Rua", che sono tutti i caratteri, il valore 15 supera la lunghezza della stringa
MsgBox strL
strR = Right(str, 7)
'Restituisce "epi Rua", che sono gli ultimi 7 caratteri (lo spazio è contato come un carattere)
MsgBox strR
strR = Right(str, 15)
'Restituisce "Bepi Rua", che sono tutti caratteri, il valore 15 supera la lunghezza della stringa
MsgBox strR
strM = Mid(str, 2, 6)
'Restituisce "epi Ru". Inizia dal secondo carattere "e" poi per 6 caratteri restituiti a partire da "e"
MsgBox strM
strM = Mid(str, 2, 15)
'Restituisce "epi Rua". Restituisce tutti i caratteri a partire dal secondo carattere "e",
' perché i caratteri specificati più il numero di partenza 2 superano la lunghezza della stringa
MsgBox strM
strM = Mid(str, 2)
'Restituisce "epi Rua". Restituisce tutti i caratteri a partire dal secondo carattere "e",
' perché il secondo argomento (char_numbers) viene omesso
MsgBox strM
strM = Mid(str, 12, 2)
'Restituisce una stringa vuota perché il numero di partenza 12 supera la lunghezza della stringa
MsgBox strM
'Restituisce 8, la lunghezza della stringa misurata dal numero di caratteri
MsgBox Len(str)
'Restituisce 8, la lunghezza della stringa misurata dal numero di caratteri.
MsgBox Len("Bepi Rua")
End Sub
```

Esempio della funzione Len con le variabili

```
Sub prova2()
'Restituisce 6 in entrambi i casi il numero di caratteri delle 2 stringhe
MsgBox Len("grande")
MsgBox Len("123456")
'Restituisce 11 il numero di caratteri della stringa incluso lo spazio
MsgBox Len("Roby Baggio")
'Una variabile di tipo variant viene trattata come una stringa
Dim vVar As Variant
vVar = 125
'restituisce 2, indicando una variabile di tip Integer
MsgBox VarType(vVar)
'Restituisce 3, il numero di caratteri contenuti nella variabile
MsgBox Len(vVar)
'variabile di tipo stringa
Dim strVar As String
strVar = "Roby Baggio"
 `ritorna 8, indicando una variabile di tipo String
MsgBox VarType(strVar)
'Restituisce 11, il numero di caratteri contenuti nella variabile di tipo String
MsgBox Len(strVar)
 `variabile di tipo integer
Dim iVar As Integer
iVar = 124
'Restituisce 2, il numero di byte utilizzati per memorizzare la variabile
MsgBox Len(iVar)
 `variabile di tipo long
Dim lVar As Long
lVar = 145
'Restituisce 4, il numero di byte utilizzati per memorizzare la variabile
MsgBox Len(lVar)
 `variabile di tipo single
Dim sVar As Single
sVar = 245.567
'Restituisce 4, il numero di byte utilizzati per memorizzare la variabile
MsgBox Len(sVar)
 `variabile di tipo double
Dim dVar As Double
dVar = 245.567
'Restituisce 8, il numero di byte utilizzati per memorizzare la variabile
MsgBox Len(dVar)
End Sub
```

Esempio delle funzioni LEFT, LEN e MID, per restituire le iniziali da una stringa di testo contenente una frase composta da più parole considerando una stringa contenente il nome, il cognome, lo stato e l'occupazione con spazi e restituire le iniziali del parole seguite da un punto e uno spazio.

```
Sub prova4()
Dim Snome As String, iniz As String, i As Integer
'Stringa di testo in cui fare la ricerca
Snome = "   Maria Stuarda regina     Scozia"
'Se il primo carattere non è uno spazio vuoto, sarà una iniziale
For i = 1 To Len(Snome)
If i = 1 Then
If Left(Snome, i) <> " " Then
iniz = Left(Snome, 1) & "."
End If
Else
'Eventuale carattere dopo il primo carattere, se è preceduto da uno spazio, sarà una iniziale
If Mid(Snome, i - 1, 1) = " " And Mid(Snome, i, 1) <> " " Then
'Per la prima iniziale
If Len(iniz) < 1 Then
iniz = Mid(Snome, i, 1) & "."
'Per più iniziali
Else
'Per più iniziali, aggiungere alla iniziale precedente
iniz = iniz & " " & Mid(Snome, i, 1) & "."
End If
End If
End If
Next i
 'Convertire tutte le iniziali in maiuscolo
iniz = UCase(iniz)
'Restituisce "M. T. R. S."
MsgBox iniz
'Restituisce 11-4 lettere, 4 punti e 3 spazi
MsgBox Len(iniz)
End Sub
```

La Funzione Replace

La funzione **Replace** in vba viene utilizzata per restituire una stringa in cui una sottostringa specificata viene sostituita, un determinato numero di volte, con un'altra sottostringa,la sintassi è la seguente:

Replace (expression, find, replace, start, count, compare)

è necessario specificare gli argomenti di *expression, find e replace*, mentre *start, count e compare* sono argomenti opzionali.

L'argomento *expression* (espressione) è la stringa che viene sostituita da una stringa specifica, si ricordi che per un valore di expression con lunghezza zero viene restituita una stringa di lunghezza zero, e per un'espressione *Null* la funzione darà un errore, mentre l'argomento *Find* specifica la stringa che deve essere sostituita e se il valore di Find è una stringa di lunghezza zero, allora viene restituita una copia di expression. L'argomento *Replace* specifica la stringa di sostituzione), notare che la stringa di sostituzione se ha lunghezza zero ha l'effetto di eliminare tutte le occorrenze dell'espressione Find e l'argomento *Start* specifica la posizione (cioè il numero di caratteri) nell'espressione da cui si desidera iniziare la ricerca della sottostringa specificata in Find.

Se questo argomento viene omesso, per impostazione predefinita assume il valore 1 (cioè la ricerca partirà dalla prima posizione del carattere) e la stringa restituita dalla funzione Replace inizia da questa posizione di partenza fino all'ultimo carattere della stringa di expression. Specificando una posizione di partenza che è maggiore della lunghezza dell'espressione, verrà restituita una stringa di lunghezza zero. L'argomento *Count* specifica il numero di sostituzioni che si desidera fare, omettendo di specificare questo valore per default assumerà il valore -1, che farà tutte le possibili sostituzioni e specificando zero per l'argomento Count avrà l'effetto di non effettuare nessuna sostituzione e restituirà una copia dell'espressione. L'argomento *Compare* specifica il tipo di confronto da utilizzare per la valutazione delle sottostringhe, può essere un valore numerico o una costante.

È possibile specificare i seguenti argomenti per l'argomento *Compare*:

- *vbUseCompareOption* (valore: -1) che esegue un confronto utilizzando l'impostazione di *Option Compare*
- *vbBinaryCompare* (valore: 0) esegue un confronto binario tra le stringhe basato su un ordinamento binario.
- *vbTextCompare* (valore: 1) esegue un confronto testuale tra le stringhe che non si basano su un ordinamento testuale case-sensitive
- *vbDatabaseCompare* (valore: 2) esegue un confronto basato sui dati di un database.

Se non si specifica l'argomento Compare, il confronto viene fatto sulla base Option definito nella dichiarazione, cioè l'istruzione Option Compare può essere *Option Compare Binary* oppure *Option Compare Text* e per essere utilizzato deve essere impostato il metodo di confronto specificando specificando 'Option Compare Binary' o 'Option Compare Text' a livello di modulo, prima di qualsiasi altra procedura. Se l'Istruzione Option Compare non è specificato, il metodo di confronto testo predefinito è Binary.

Esempio: Come utilizzare la funzione Replace

```
Sub sostituisci1()
    Dim str As String, strF As String, strR As String
'Trovare tutte le occorrenze di "a" da sostituire
strF = "s"
strR = ""
str = "Laura non c'è è Andata via!"
'Restituisce 27
MsgBox Len(str)
'Restituisce la stringa dopo l'eliminazione di tutte le occorrenze di' a', notare che
'il valore A non viene rimosso
str = Replace(str, strF, strR)
MsgBox str
'Restituisce 22
MsgBox Len(str)
str = "Laura non c'è è Andata via!"
'Restituisce 27
MsgBox Len(str)
'Restituisce la stringa dopo l'eliminazione di tutte le occorrenze di' a ' notare che anche il
carattere 'A'
' anche il carattere 'A' viene sostituito perché il valore di confronto è vbTextCompare
str = Replace(str, strF, strR, , , 1)
MsgBox str
MsgBox Len(str)
'Restituisce 21
str = "Laura non c'è è Andata via!"
'Restituisce 27
MsgBox Len(str)
'Eliminazione di tutte le occorrenze di' a ' notare che il numero di caratteri
'6 è uno spazio vuoto e viene restituito.
'La stringa restituita dalla funzione inizia dalla posizione di partenza fino all'ultimo
'carattere della stringa di espressione.
str = Replace(str, strF, strR, 6)
MsgBox str
'Restituisce 16
MsgBox Len(str)
str = "Laura non c'è è Andata via!"
'Restituisce 27
MsgBox Len(str)
'Specifica una posizione di partenza che è maggiore della lunghezza dell'espressione,
'restituirà una stringa di lunghezza zero.
str = Replace(str, strF, strR, 28)
'Restituisce una stringa di lunghezza zero
MsgBox str
'restituisce 0:
MsgBox Len(str)
str = "Laura non c'è è Andata via!"
'Restituisce 27
MsgBox Len(str)
str = Replace(str, strF, strR, 8, 2)
MsgBox str
MsgBox Len(str)
End Sub
```

Esempio: Sostituire tutte le occorrenze di una stringa in un'espressione stringa con un'altra stringa

```
Function sostituisci2(var As Variant, varF As Variant, varR As Variant, optI As Integer) As
Variant
   Dim posF As Integer
'Posizione della prima occorrenza di varFind, entro var
posF = InStr(var, varF)
'Se posF non viene trovato all'interno var
If posF < 1 Then
'restituisce var
sostituisci2 = var
Else
'Sostituire tutte le istanze di varF
sostituisci2 = Replace(var, varF, varR, , , optI)
End If
End Function

Sub cambia()
Dim var As Variant, varF As Variant, varR As Variant, optI As Integer
'var è la stringa all'interno della quale varF viene cercato e sostituito
var = "Laura non c'è è Andata via!"
'varF è la stringa da cercare all'interno di var e che sarà sostituito
varF = "a"
'varR è la stringa che sostituisce tutte le istanze di varF entro var
varR = "?"
'Per eseguire un confronto binario (case-sensitive),  si utilizza il valore 0 (optI=0)
'Per eseguire un confronto di testo (case insensitive), si utilizza il valore 1
optI = 1
'Se var è Null, si esce
If IsNull(var) Then
MsgBox "Var è nullo, esco dalla procedura"
Exit Sub
'Se var non è Null
Else
'Se uno fra varF o varR sono Null o varFè una stringa di lunghezza zero
If IsNull(varF) Or IsNull(varR) Or varF = "" Then
'Ritorno var senza sostituzioni e esco dalla procedura
MsgBox var
Exit Sub
Else
'Se var, varF, varR non sono Null, eseguo la funzione di sostituire tutte le istanze di varF
MsgBox sostituisci2(var, varF, varR, optI)
End If
End If
End Sub
```

Le Funzioni Split, Join, InStr e InStrRev

La Funzione Split

La funzione **Split** divide una stringa in un numero specificato di sottostringhe e usando un carattere separatore in essa inclusa restituisce tutte le sottostringhe di cui la stringa originale è composta in una matrice unidimensionale in base zero. Sintassi:

Split (expression, delimiter, limit, compare).

È necessario specificare solo l'argomento *expression* mentre tutti gli altri argomenti sono facoltativi.
L'argomento expression è la stringa che verrà divisa in sotto stringhe e delimitata da un carattere contenuto all'interno. Per una stringa di lunghezza zero ("") la funzione restituisce un array vuoto senza elementi.
L'argomento *delimiter* è il carattere utilizzato per delimitare e separare le sottostringhe e identifica i limiti delle sottostringhe, e se viene omesso, verrà assunto il carattere spazio ("") di default come delimitatore, mentre se è una stringa di lunghezza zero (""), la funzione restituisce l'intera espressione come una matrice a elemento singolo.
L'argomento *limit* specifica il numero di sottostringhe da restituire e il valore di -1 indica che tutte le sottostringhe vengano restituite.
L'argomento *compare* specifica il tipo di confronto da utilizzare per valutare le stringhe

È possibile specificare i seguenti argomenti per l'argomento compare:

- *vbUseCompareOption* (valore: -1) esegue un confronto utilizzando l'impostazione di Option Compare.
- *vbBinaryCompare* (valore: 0) esegue un confronto binario - confronti tra stringhe basato su un ordinamento
- *vbTextCompare* (valore: 1) esegue un confronto testuale - confronti tra stringhe che non si basano su un ordinamento testuale case-sensitive
- *vbDatabaseCompare* (valore: 2) esegue un confronto basato sui dati del database

Se non si specifica l'argomento compare, il confronto viene fatto sulla base Option Compare definita *Option Compare Statement*, cioè Option Compare Binary oppure Option Compare Text che può essere utilizzato per impostare il metodo di confronto che è necessario specificare a livello di modulo, prima di qualsiasi procedura e se l'Istruzione Option Compare non è specificata, il metodo di confronto testo predefinito è Binary.

Esempio: Contare e ritornare le parole all'interno di una stringa

Fig. 1

```
Sub Split2()
Dim testo1 As Variant, varE As Variant, varD As Variant, varP As Variant
Dim i As Integer
'stringa che sarà suddivisa in sotto stringhe - ogni parola è separata da uno spazio
varE = "    Ho visto    un    re anche lui    piangeva.  "
varD = " " 'Indicare lo spazio come delimitatore
'con TRIM si rimuovono tutti gli spazi dal testo ad eccezione dei singoli spazi tra le parole
varE = Application.Trim(varE)
testo1 = Split(varE, varD)
'Restituisce il numero di parole (7) nella stringa
MsgBox UBound(testo1) + 1
'mettere ogni parola della stringa su righe diverse
For i = 0 To UBound(testo1)
If i = 0 Then
varP = testo1(i)
Else
varP = varP & vbLf & testo1(i)
End If
Next i
MsgBox varP 'Restituisce ogni parola in una riga separata
End Sub
```

Esempio: Estrarre le sotto stringhe utilizzando la funzione Split e riportare la lunghezza e il numero di occorrenze del carattere delimitatore all'interno di una stringa

```
Sub split1()
Dim test As Variant, varE As Variant, varD As Variant
Dim i As Integer, lungE As Long, lungEx As Long
varE = "le belle vie del paese" 'stringa che sarà suddivisa in sotto stringhe
varD = "e" 'delimitatore della stringa
test = Split(varE, varD)
'Restituisce il numero di elementi nella matrice
MsgBox UBound(test) + 1
'Restituisce 8, il n° di occorrenze del delimitatore all'interno della stringa
MsgBox UBound(test)
For i = LBound(test) To UBound(test)
'riporta ogni elemento della matrice in cui la stringa è divisa
MsgBox test(i)
'ritorna la lunghezza di ogni elemento della matrice
MsgBox Len(test(i))
'riporta la lunghezza totale di tutti gli elementi della matrice
lungE = lungE + Len(test(i))
Next i
lungEx = Len(varE) 'lunghezza della stringa divisa
'calcolare la lunghezza di expression
If lungEx = UBound(test) * Len(varD) + lungE Then
MsgBox "Uguale"
Else
MsgBox "Diverso"
End If
End Sub
```

Esempio: Estrarre un elemento di un array, il nome del sito da un indirizzo web o il nome del file.

```
Sub split3()
Dim testo1 As Variant, varE As Variant, varD As Variant
Dim n As Integer
varE = "E la luna bussò alle porte del sole" 'stringa da estrarre
varD = " " 'Indicare lo spazio come delimitatore di stringhe
 testo1 = Split(varE, varD)
n = 3 'Estrarre il terzo elemento della stringa precedente
MsgBox testo1(n - 1)
MsgBox Split("22,456,7,9824,0", ",")(n - 1)
varE = http://forum.wintricks.it/showthread.php?t=155252 'indicare sito web
varD = "/"
testo1 = Split(varE, varD)
n = 3
MsgBox testo1(n - 1)
varE = "C:\User\Alex\Documents\Excel\VBA\#39.xls" 'Specificare il percorso di un file
varD = "\"
testo1 = Split(varE, varD)
n = UBound(testo1) + 1 'Estrarre l'ultimo elemento - il nome del file
MsgBox testo1(n - 1)
MsgBox testo1(UBound(testo1))
End Sub
```

Esempio: Sostituire tutte le occorrenze di una stringa in un'espressione stringa con un'altra stringa

```
Function Rep_1(var As Variant, varF As Variant, varR As Variant, opt1 As Integer) As Variant
   Dim contaF As Integer, arr As Variant
arr = Split(var, varF, , opt1)
If UBound(arr) < 1 Then 'Se varF non è stato trovato in var l'array avrà un solo elemento
Rep_1 = var 'Ritorno la stringa varr ed esco dalla procedura
Exit Function
Else
var = ""  'Inizio con una stringa di lunghezza zero
For contaF = 1 To UBound(arr) 'Ciclo per il n° di occorrenze
var = var & arr(contaF - 1) & varR 'Aggiunge ogni elemento (tranne l'ultimo) della matri. con varR
Next contaF
var = var & arr(UBound(arr)) 'Aggiungere l'ultimo elemento dell'array dopo tutte le sostituzioni
End If
Rep_1 = var 'Ritorno la stringa finale
End Function
Sub cambia1()
Dim var As Variant, varF As Variant, varR As Variant, opt1 As Integer
varR = "?" 'var è la stringa all'interno della quale varF viene cercato e sostituito da varR
opt1 = 1 'valore per eseguire il confronto di testo
If IsNull(var) Then 'Se var è Null, si esce
MsgBox "var è nullo, esco dalla procedura"
Exit Sub
Else 'Se var non è Null
If IsNull(varF) Or IsNull(varR) Or varF = "" Then
MsgBox var
Exit Sub
Else
MsgBox Rep_1(var, varF, varR, opt1)
End If
End If
End Sub
```

La Funzione Join

La funzione **Join** unisce le sottostringhe contenute in una matrice, e restituisce una stringa con le sottostringhe separate da un carattere delimitatore. Sintassi:

Join (sourceArray, delimiter).

È necessario specificare l'argomento sourceArray mentre l'argomento delimiter è facoltativo, ricordare che *sourceArray* è un array che contiene le sottostringhe che devono essere unite per restituire una stringa e *delimiter* è il carattere stringa utilizzato per separare le sottostringhe, e se viene omesso, verrà assunto il carattere di spazio ("") di default per essere usato come delimiter, se invece delimiter è una stringa di lunghezza zero (""), la funzione unisce le stringhe senza delimitatore.

Esempio: Utilizzo della funzione JOIN

```
Sub join1()
Dim arr As Variant, varJ As Variant, varC As Variant
Dim i As Integer
'Definire l'array
arr = Array("America", "Europa", "Africa", "Asia")
 'unire sottostringhe contenute in una matrice
varJ = Join(arr, "&")
'Ritorna la stringa unita
MsgBox varJ
'Concatenare ogni elemento della matrice
For i = 0 To UBound(arr)
varC = varC & "&" & arr(i)
Next i
'Rimuovere la "&" prima del primo elemento
varC = Mid(varC, 2)
'String ritorno dopo il concatenamento
MsgBox varC
End Sub
```

Esempio: Unire i valori delle celle in un intervallo del foglio di lavoro

	A	B	C	D	E
1	Nome	Città	Età	Sesso	Stato
2	Gino Primo	Milano	50	M	Celibe
3	Marco Secondo	Roma	42	M	Sposato
4	Teresa Terzo	Venezia	36	F	Sposata
5	Elena Quinto	Palermo	75	F	Celibe

Fig. 2

```
Sub join2()
    Dim rng As Range, riga1 As Integer, colonna1 As Integer, i As Integer
Set rng = ActiveSheet.Range("A2:E4")
Dim varC As Variant
For riga1 = 1 To rng.Rows.Count
For colonna1 = 1 To rng.Columns.Count
If Not rng(riga1, colonna1).Value = vbNullString Then
varC = varC & "," & rng(riga1, colonna1).Value
End If
Next colonna1
'Se l'array è vuoto
If varC = vbNullString Then MsgBox "Array Vuoto": GoTo skip1
'restituisce un record per riga
MsgBox Mid(varC, 2)
varC = ""

skip1:
Next riga1
'Dichiarare una matrice dinamica
Dim varA() As Variant
Icoll = rng.Columns.Count
i = 0
'Ridimensionare la matrice dinamica
ReDim varA(Icoll - 1) As Variant
 For riga1 = 1 To rng.Rows.Count
For Icoll = 1 To rng.Columns.Count
If Not rng(riga1, Icoll).Value = vbNullString Then
'Per ogni vbNullString, diminuire il valore dell'indice di matrice
varA(Icoll - 1 - i) = rng(riga1, Icoll).Value
Else
'Contare il numero di vbNullString
i = i + 1
'Se l'array è vuoto
If i = rng.Columns.Count Then MsgBox "Array Vuoto": GoTo skip2
End If
Next Icoll
'Diminuire la dimensione della matrice per numero di vbNullString
ReDim Preserve varA(rng.Columns.Count - 1 - i) As Variant
'restituisce un record per riga
MsgBox Join(varA, ",")

skip2:
'Diminuire la dimensione della matrice per numero di vbNullString
ReDim varA(rng.Columns.Count - 1) As Variant
i = 0
Next riga1
End Sub
```

Esempio: Usare la funzione Split

```
Sub split2()
Dim newT As Variant, varS As Variant, varD As Variant, varE As Variant, varJ As Variant
Dim i As Integer
varS = http://forum.wintricks.it/showthread.php?t=155252  'indirizzo web
varD = "/" 'delimitatore
newT = Split(varS, varD) 'Restituisce un array
For i = 0 To UBound(newT) `mettere ogni elemento della matrice su una riga separata
If i = 0 Then
varE = newT(i) 'Non inserire una interruzione di linea prima del primo elemento
Else
varE = varE & vbLf & newT(i)
End If
Next i
MsgBox varE 'Restituisce ogni elemento su una riga separata
varJ = Join(newT, varD) 'restituisce l'espressione stringa originale
MsgBox varJ
End Sub
```

Esempio: Utilizzo delle funzioni stringa

```
Sub demo1()
    Dim newT As Variant, varE As Variant, varSE As Variant, varD As Variant, varJ As Variant
Dim Nfile As String, Fdir As String
 'stringa da cui si desidera estrarre un elemento
varE = "Estrarre una sotto espressione dopo aver escluso un elemento da un'espressione"
varD = " "
 `restituire una matrice unidimensionale in base zero
newT = Split(varE, varD)
'Escludere un elemento assegnare il n° dell'elemento (2) a una variabile
n = 2
For i = 0 To UBound(newT)
If i = n - 1 Then
varSE = varSE
Else
varSE = varSE & "," & newT(i)
End If
Next i
'Rimuovere il primo ","
varSE = Mid(varSE, 2)
MsgBox varSE
'Ridimensionare la matrice per ridurre gli elementi di 1 in modo da escludere l'ultimo elemento
ReDim Preserve newT(UBound(newT) - 1)
'Unire tutti gli elementi dell'array tranne l'ultimo e aggiungere il . alla fine
varJ = Join(newT, varD) & "."
MsgBox varJ 'Stringa estratta, escluso l'ultimo elemento
varE = "C:\User\Alex\Documents\Excel\VBA\pippo.xls" `indicare il percorso del file
Nfile = Mid(varE, InStrRev(varE, "\") + 1) 'Estrarre il nome del file, dal percorso completo del file
MsgBox Nfile
Fdir = Left(varE, Len(varE) - Len(Nfile)) 'Estrarre il percorso della cartella, escluso il nome del file
MsgBox Fdir
End Sub
```

La Funzione InStr e InStrRev

La funzione **InStr** restituisce la posizione (numero di caratteri) in cui una stringa prima si verifica all'interno di un'altra stringa. Sintassi:

InStr (start, string, substring, compare)

ed è necessario specificare gli argomenti *string* e *substring*, mentre gli argomenti *start* e *compare* sono opzionali.

L'argomento *start* specifica la posizione (numero di caratteri) all'interno della stringa da cui si desidera iniziare la ricerca per *substring*, è necessario specificare l'argomento start, se l'argomento di confronto è da specificare e se viene omesso, per impostazione predefinita assumerà il valore 1 (cioè la ricerca partirà dalla prima posizione del carattere).

Specificando una posizione di partenza che è maggiore della lunghezza di string verrà restituito il valore 0 (zero), e se start contiene un valore *Null* si verificherà un errore. L'argomento *string* è l'espressione stringa all'interno della quale cercare *substring*, la funzione restituisce 0 se la stringa è di lunghezza zero, e restituisce Null se la stringa è Null. L'argomento *substring* è l'espressione stringa che viene cercata all'interno della stringa e la cui posizione verrà restituito dalla funzione che restituisce 0 se stringa non viene trovata, oppure restituisce il valore iniziale se la stringa è di lunghezza zero, o restituisce Null se la stringa è Null. L'argomento *compare* specifica il tipo di confronto da utilizzare per valutare le stringhe.

È possibile specificare i seguenti argomenti per l'argomento compare:

- *vbUseCompareOption* (valore: -1) esegue un confronto utilizzando l'impostazione di Option Compare.
- *vbBinaryCompare* (valore: 0) esegue un confronto binario
- *vbTextCompare* (valore: 1) esegue un confronto testuale - confronti tra stringhe che non si basano su un ordinamento testuale case-sensitive
- *vbDatabaseCompare* (valore: 2) esegue un confronto basato sui dati del database

Se non si specifica l'argomento *compare* , il confronto viene fatto sulla base Option definito nella dichiarazione, ricordare che l'istruzione Option Compare (cioè Option Compare Binary o Option Compare Text) può essere utilizzato per impostare il metodo di confronto ed è necessario specificare 'Option Compare Binary' o 'Option Compare Text' a livello di modulo, prima di qualsiasi altra procedura. Se l'Istruzione Option compare non è specificata, il metodo di confronto predefinito è Binary.

La **funzione InStrRev** Restituisce la posizione della prima occorrenza di una stringa inclusa in un'altra a partire dalla destra della stringa con la seguente sintassi:

InStrRev (string, substring, start, compare)

mentre si utilizzare la funzione *InStrRev* invece di InStr per cercare nella direzione opposta. È necessario specificare gli argomenti di stringa e sottostringa, mentre gli argomenti start e compare sono opzionali. Se viene omesso l'argomento start, viene utilizzato -1, che significa che la ricerca inizierà dalla posizione dell'ultimo carattere. Tutte le altre spiegazioni e sintassi rimangono invariati rispetto alla funzione InStr.

Esempio: Utilizzo della funzione InStr.

```
Sub InStrFunc()
    Dim str1 As String, str2 As String
str1 = "Alice vince sempre"
str2 = "e"
MsgBox InStr(str1, str2) 'restituisce 5
str1 = "Alice vince sempre"
str2 = "e"
MsgBox InStr(4, str1, str2) 'restituisce 5
 str1 = "Alice vince sempre"
str2 = "e"
MsgBox InStr(24, str1, str2) 'restituisce 0
str1 = ""
str2 = "e"
MsgBox InStr(str1, str2) 'restituisce 0
 str1 = "Alice vince sempre"
str2 = "i"
MsgBox InStr(str1, str2) 'restituisce 3
Dim str3 As Variant
str1 = "Alice vince sempre"
str3 = Null
MsgBox VarType(InStr(str1, str3)) 'restituisce 1
str1 = "Alice vince sempre"
str2 = "s"
MsgBox InStr(2, str1, str2) 'restituisce 13
MsgBox InStr(2, str1, str2, 1) 'restituisce 13
End Sub
```

Esempio: Sostituire tutte le occorrenze di una stringa in un'espressione stringa con un'altra stringa

```
Function cambia1(var As Variant, varF As Variant, varR As Variant) As Variant
Dim Lfind As Integer, Pfind As Integer
Pfind = InStr(var, varF) 'Posizione della prima occorrenza di varF, in var
Lfind = Len(varF) 'Lunghezza varF, che sarà sostituito con varR
RPlen = Len(varR) 'Lunghezza RPlen, che sarà sostituito da varF
'Se varF non viene trovato all'interno var
If Pfind < 1 Then
'si restituisce la stringa var stringa e si esce dalla  procedura
cambia1 = var
Exit Function
'Se varF viene trovato all'interno di var
Else
Do
'Sostituire varF con varR in var
var = Left(var, Pfind - 1) & varR & Mid(var, Pfind + Lfind)
'Posizione della prima occorrenza di varF all'interno di var aggiornato, iniziando
' la ricerca dal primo carattere dopo l'ultima sostituzione
Pfind = InStr(Pfind + RPlen, var, varF)
'Se varF non è stato trovato all'interno aggiornato di var, si esce dal ciclo
If Pfind = 0 Then Exit Do
Loop
End If
cambia1 = var 'Ritorno stringa finale
End Function

Sub cambia2()
Dim var As Variant, varF As Variant, varR As Variant
'var è la stringa all'interno della quale varF viene cercato e sostituito da varR
 var = "Alice vince sempre"
'varF è la stringa da cercare all'interno di var e che sarà sostituito da varR
varF = "e"
'varR è la stringa che sostituisce tutte le istanze di varF contenute in var
varR = "?"
 'Se var è Null, si esce
If IsNull(var) Then
MsgBox "var è Null, esco dalla procedura"
Exit Sub
'Se var non è Null
Else
'Se uno tra varF o varR sono Null o varF è una stringa di lunghezza zero
If IsNull(varF) Or IsNull(varR) Or varF = "" Then
'Ritorno var senza sostituzioni e si esce dalla procedura
MsgBox var
Exit Sub
Else
'Se nessuno tra var, varF e varR sono Null, si esegue la funzione e si sostituiscono tutte le istanze
di varF
MsgBox cambia1(var, varF, varR)
End If
End If
End Sub
```

Le Matrici - Statiche e Dinamiche

Una **Matrice** (o Array) è una collezione di variabili che condividono lo stesso nome e tipo di dati e costituiscono un comodo metodi per memorizzare un certo numero di dati nello stesso contenitore. Diversamente da quanto accade con i tipi di dati definiti dall'utente, gli elementi di una matrice devono essere *tutti dello stesso tipo*. Per esempio se si crea una matrice di tipo Integer, tutti gli elementi al suo interno devono essere di tipo Integer, in altre parole le matrici vengono utilizzate per rappresentare elenchi o tabelle in cui tutti i dati siano dello stesso tipo. Una matrice permette di memorizzare ed elaborare una serie di dati utilizzando una sola variabile che permette oltre alla riduzione complessiva di nomi di variabili da gestire, la possibilità di utilizzare dei cicli per semplificare l'elaborazione dei diversi elementi che la compongono.

Combinando matrici e cicli (tipicamente cicli *For Next* e *ForEach*) è possibile scrivere procedure di poche istruzioni che elaborano grandi quantità di dati, che se si dovesse usare nomi distinti per le variabili, le stesse procedure potrebbero richiedere centinaia di istruzioni. La forma più semplice di una matrice non è altro che un elenco di elementi che vengono definite semplici o unidimensionali

Dichiarazioni di Matrici

Per dichiarare una matrice è necessario prevedere il limite superiore (l'ultimo e più grande numero di indice), mentre il limite inferiore (il primo e il più piccolo numero di indice) è facoltativo e se si omette questo argomento, viene determinato dalle impostazione del modulo, che di default è uguale a 0. Per riuscire a tenere traccia delle dimensioni di un array siano essi statici o dinamici il VBA prevede due funzioni, *Lbound* e *Ubound*, che restituiscono il valore minimo e massimo per gli indici di un array. La sintassi generica per queste funzioni è

LBound(NomeArray [, dimensione])

UBound(NomeArray [, dimensione])

L'argomento "*dimensione*" è un numero intero che specifica per quale dimensione dell'array si vuole ottenere il limite minimo o massimo. Se non viene specificato VBA restituisce l'estremo relativo alla prima dimensione dell'array. La funzione "*UBound*" restituisce il valore della posizione più alta per la dimensione indicata di una matrice, mentre "*LBound*" è l'opposto e restituisce il valore più basso possibile, mentre il valore di ritorno per entrambe queste funzioni è un dato di tipo Integer. Quando un array utilizza un solo indice viene chiamato "*unidimensionale*", mentre una matrice "*multidimensionale*" utilizza più di un indice.

Va ricordato che la dimensione di una matrice è il numero totale di elementi che è determinato dal prodotto delle sue dimensioni e per determinare la sua dimensione massima si deve tenere presente che gli indici partono da 0 e il valore massimo è definito dal numero indicato nella dichiarazione. È possibile dichiarare un array multidimensionale di tipo Byte con la sintassi: Dim K As Byte e ottenere la dimensione della matrice utilizzando la funzione "UBound", oppure utilizzare la funzione LBound per ottenere il valore più basso possibile per l'indice. Se l'array ha un solo elemento, il valore di ritorno di Ubound è uguale a 0 e LBound restituisce sempre 0 finchè l'array non è stato inizializzato, anche se non ha elementi. Esempio per un array monodimensionale: *Dim MyArray1 (5) As String*

Limite Inferiore = Lbound (MyArray1) 'Restituisce il valore 0
Limite Superiore = Ubound (MyArray1) 'Restituisce il valore 0
Dimensioni = Ubound (MyArray1) - Lbound (MyArray1) + 6 'Restituisce il valore 6

Si deve tenere presente che quando si utilizzano queste funzioni per un array multi-dimensionale, si deve specificare la dimensione in base al quale il limite inferiore o superiore devono essere determinati.

Dovrebbe risultare familiare l'uso dell'istruzione **Dim** che abbiamo visto usare per le variabili, ma viene usata anche per la dichiarazione di matrici, in effetti la parola chiave Dim è un'abbreviazione di *dimension* e nella versione moderna del VBA la parola chiave *Dim* permette di dichiarare matrici sia unidimensionali che a più dimensioni. La sintassi generica per la dichiarazione è la seguente

Dim Nome_variabile ([Indici]) [As Tipo]

Nome_variabile rappresenta un qualsiasi nome per la matrice che risponda alle regole di VBA per i nomi delle variabili e l'argomento *Indici* rappresenta le dimensioni della matrice, che è possibile dichiarare con un numero di dimensioni fino a 60. Per una matrice unidimensionale si usa una sola dichiarazione di indici, per una a due dimensioni se ne includono due separate da una virgola e via così fino a raggiungere il numero di dimensioni desiderato per la matrice. La sezione Indici ha la seguente sintassi

[minimo To] massimo [, [minimo To] massimo]

Minimo specifica il valore minimo valido per l'indice della matrice e *massimo* quello più alto, si deve tenere presente che è richiesto soltanto il limite superiore, in quanto l'indicazione del valore minimo per l'indice è opzionale. Se si specifica solo il limite superiore, VBA numera gli elementi della matrice coerentemente con l'impostazione *Option Base*, se è stato specificato *Option Base 1*, VBA numera gli elementi della matrice da 1 al valore massimo, in caso contrario gli elementi della matrice vengono numerati da 0 al valore massimo. Per esempio

Dim gennaio (1 to 31) As String
Dim gennaio(31) As String

Nella seconda riga del listato poco sopra si presume *Option Base 1* e specificando la parte *minimo To* si facilita la comprensibilità del codice e si semplifica la ricerca di errori, rendendo i programmi più affidabili. Dichiarare la parte *minimo To* permette anche di specificare come valore di partenza per l'indice un valore diverso da 0 o 1. Per esempio potreste voler creare una matrice i cui elementi siano numerati da 5 a 10 o da -5 a 0, a seconda della particolare operazione che intendete compiere. Come nelle normali dichiarazioni di variabili, è possibile dichiarare per una matrice un particolare tipo di dati specificando la parte *As tipo*, in questo modo ogni elemento della matrice sarà del tipo specificato, che può essere qualsiasi tipo di dato VBA valido

Finora abbiamo visto matrici dotate di indici che iniziano da 0 (*zero-based*), che secondo questa convenzione di numerazione, l'indice del primo elemento di qualsiasi dimensione di una matrice è 0, una matrice di 5 elementi ha dunque indici che vanno da 0 a 4. Evidentemente la numerazione *zero-based* può generare confusione perché l'elemento di indice zero in realtà indica il primo componente della matrice l'indice 4 indica il quinto elemento e così via. Sarebbe molto più comodo se gli elementi di una matrice fossero numerati a partire da 1 invece che da 0. In questo caso l'indice 1 indicherebbe il primo elemento della matrice, l'indice 5 il quinto e così via. Fortunatamente VBA permette di specificare il numero iniziale per gli elementi di una matrice, infatti è possibile specificare se si desidera che gli indici della matrice partano da 0 o da 1 con l'istruzione **Option Base** che presenta questa sintassi

Option Base 0|1

L'istruzione Option Base permette di impostare 0 o 1 come indice di base della matrice, in mancanza di questa istruzione VBA assegna per default la numerazione degli elementi della matrice a partire da 0. L'istruzione Option Base deve essere inserita nell'area delle dichiarazioni di un modulo prima di qualsiasi dichiarazione di variabile, costante o procedura e non può comparire all'interno di una procedura. Ecco due esempi

Option Base 0 'Impostazione predefinita
Option Base 1 'Gli indici della matrice partono da 1

Matrici Unidimensionali

Un array monodimensionale è un elenco di elementi che hanno una singola riga o una singola colonna, ad esempio, le vendite trimestrali di una società durante l'anno (la dimensione sarà la riga o la colonna dei 4 trimestri dell'anno per il quale i dati di vendita saranno inseriti). Di seguito vediamo alcune dichiarazioni per creare un array:

Per creare un array con 7 elementi, con i numeri di indice 0-6 (il valore predefinito del limite inferiore è 0) si usa la sintassi: *Dim prova1(6) As String* oppure *Dim prova1(0 To 6) As String*. In questo secondo caso il limite inferiore deve essere specificato in modo esplicito, con la parola chiave *To*. Supponiamo di avere un array che ha 20 elementi in cui i numeri di indice vanno da 1 a 20 si può usare la seguente dichiarazione: *Dim prova2(1 To 20) As Integer*.

Nel caso si abbia un array con 51 elementi in cui i numeri di indice vanno da 100 a 150 si può usare la seguente dichiarazione: *Dim prova2(100 To 150) As Integer*.
In sostanza i formati per una dichiarazione di matrice unidimensionale, con e senza specificare il limite inferiore possono essere rispettivamente:

Dim ArrayName(Index) as DataType, oppure
Dim ArrayName(First_Index To Last_Index) as DataType

Pertanto possiamo riassumere che il nome è legato al fatto che un elenco di dati è assimilabile a una linea tracciata su un foglio, che ha una sola dimensione, e una sola lunghezza, ed è quindi *unidimensionale* o monodimensionale.

10	Elemento 0
10,5	Elemento 1
11	Elemento 2
11,6	Elemento 3
11,9	Elemento 4

Fig. 1

La **Figura 1** rappresenta una matrice *unidimensionale* e ogni dato memorizzato nella matrice è detto elemento della matrice e come si vede in figura la matrice contiene 5 elementi, ciascuno dei quali contiene un numero di tipo *Double*, noterete anche che gli elementi della matrice sono numerati da 0 a 4 per un totale di 5 e che la numerazione parte da zero. Per accedere al dato memorizzato in un certo elemento si usa il nome della matrice seguito dal numero (detto indice) dell'elemento desiderato racchiuso tra parentesi tonde. Per esempio se il nome della matrice fosse *NumMat* la seguente istruzione assegnerebbe il valore 11,6 alla variabile *alex*

alex = NumMat(3)

Nell'esempio sopra riportato *3 è l'indice della matrice*, è racchiuso tra parentesi tonde e non è separato da spazi dal nome della matrice. Visto che la numerazione degli elementi parte da zero, l'elemento a cui fa riferimento l'istruzione è in realtà il numero 4 di *NumMat* e quando si esegue questa istruzione VBA legge il valore 11,6 dall'elemento della matrice indicato e lo memorizza nella variabile alex come per qualsiasi assegnamento.

L'indice viene anche utilizzato tutte le volte che si vuole salvare un valore in un certo elemento della matrice, l'istruzione seguente, per esempio, memorizza nel secondo elemento della matrice il valore 12

NumMat(2) = 12

Quando VBA esegue l'istruzione sopra riportata, scrive il valore 12 nell'elemento della matrice indicato, sostituendone il contenuto precedente, esattamente come per qualsiasi assegnazione, potete usare gli elementi di una matrice all'interno di una espressione VBA come se si trattasse di una qualsiasi variabile. Vediamo degli esempi di codice per un array monodimensionale:

```
Sub demo1()
Dim K1(3) As String
K1(0) = "Primo "
K1(1) = "Secondo "
K1(2) = "Terzo "
K1(3) = "Quarto "
MsgBox K1(0) & "- " & K1(1) & "- " & K1(2) & "- " & K1(3)
End Sub
```

E riporta un messaggio come il seguente:

Fig. 2

```
Sub demo2()
Dim K2(-3 To 2) As String
K2(-3) = "Uno"
K2(-2) = "Due"
K2(-1) = "Tre"
K2(0) = "Quattro"
K2(1) = "Cinque"
K2(2) = "Sei"
MsgBox K2(-3) & " - " & K2(-2) & " - " & K2(-1) & " -" & K2(0) & " - " & K2(1) & " - " &
K2(2)
End Sub
```

E riporta un messaggio come il seguente:

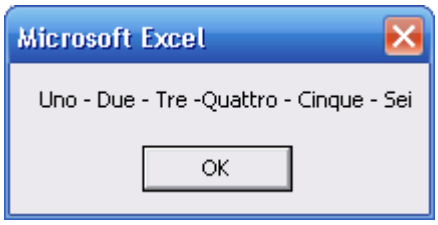


Fig. 3

Esempio: gestire gli array monodimensionali con un ciclo

	A	B
1	Nome	Peso
2	Mario	85 Kg.
3	Piero	96 Kg.
4	Alice	47 Kg.
5	Rosa	45 Kg.

Fig. 4

```
Sub demo3()
Dim i As Integer
Dim nome(2 To 5) As String, peso(2 To 5) As Single
For i = 2 To 5
nome(i) = InputBox("Inserisci il Nome")
peso(i) = InputBox("Inserisci il Peso")
Sheet4.Cells(i, 1) = nome(i)
Sheet4.Cells(i, 2) = peso(i) & " Kg."
Next i
End Sub
```

Matrici Multidimensionali

Le matrici unidimensionali vanno bene finchè si tratta di rappresentare semplici elenchi di dati, spesso però nei vostri programmi vi troverete a dover rappresentare tabelle di informazioni in cui i dati sono organizzati in righe e colonne, più o meno come accade in un foglio di Excel, per farlo dovete ricorrere a una matrice multidimensionale, che rappresentiamo in **figura 5**

	Colonna 0	Colonna 1	
	10	12,4	Elemento 1,0
Elemento 0,1	10,5	15,2	
	11	18	
Elemento 0,3	11,6	13	Elemento 1,3
	11,9	19,8	

Fig. 5

Una matrice bidimensionale ha 2 dimensioni, comprendenti righe e colonne e utilizza due indici, uno rappresenta le righe e l'altro rappresenta le colonne e vengono utilizzati quando è necessario specificare un elemento con due attributi, ad esempio, le vendite trimestrali di una società nel corso degli ultimi 5 anni (una dimensione saranno i 4 trimestri dell'anno e la seconda dimensione saranno i 5 anni). Un array bidimensionale appare con una forma a tabella, con più righe e colonne e la sintassi è:

Dim ArrayName (Index1, Index2) As DataType oppure
Dim ArrayName (First_Index1 To Last_Index1, First_Index2 To Last_Index2) As DataType.

Per esempio se si dichiara un array bidimensionale si utilizza la seguente forma:

Dim prova (7,9) As Long oppure *Dim prova (7,1 To 9) As Long*

Tenendo presente che il limite inferiore può essere specificato in modo esplicito in una o entrambe le dimensioni.

Semplificando quanto esposto possiamo affermare che le matrici *multidimensionali* devono il loro nome al fatto di avere più di una dimensione, la lunghezza (il numero di righe) e la larghezza (il numero di colonne), come si vede in in *figura 5*, la matrice ha due colonne (numerate 0 e 1) e 5 righe numerate da 0 a 4 per un totale di 10 elementi. Per accedere agli elementi di matrici multidimensionali si usa un indice, cioè si usano i riferimenti di riga e colonna per identificare un certo elemento. L'indicizzazione di una matrice bidimensionale assomiglia molto al metodo per identificare le celle di un foglio di lavoro di Excel, la prima dimensione della matrice corrisponde alle colonne del foglio di lavoro e la seconda alle righe. Se chiamiamo *NewMat* la matrice di *figura 5* la seguente istruzione assegna alla variabile alex il valore 12,4 (prima riga, seconda colonna della matrice)

alex = NewMat(1,0)

Analogamente, la seguente istruzione memorizza il valore 5,5 nella seconda riga della prima colonna della matrice

NewMat(0,1) = 5,5

Da notare che in entrambe le precedenti istruzioni gli indici della matrice sono racchiuse tra parentesi e che le coordinare della colonna e della riga sono separate da virgole. Le matrici possono avere più di due dimensioni, la figura sottostante (**Fig. 6**) mostra una matrice a tre dimensioni con una lunghezza, una larghezza e una profondità (per modo di dire)

0		1		2	
Colonna 0	Colonna 1	Colonna 0	Colonna 1	Colonna 0	Colonna 1
Elemento 0,0,0	Elemento 0,1,0	Elemento 1,0,0	Elemento 1,1,0	Elemento 2,0,0	Elemento 2,1,0
Elemento 0,0,1	Elemento 0,1,1	Elemento 1,0,1	Elemento 1,1,1	Elemento 2,0,1	Elemento 2,1,1
Elemento 0,0,2	Elemento 0,1,2	Elemento 1,0,2	Elemento 1,1,2	Elemento 2,0,2	Elemento 2,1,2
Elemento 0,0,3	Elemento 0,1,3	Elemento 1,0,3	Elemento 1,1,3	Elemento 2,0,3	Elemento 2,1,3
Elemento 0,0,4	Elemento 0,1,4	Elemento 1,0,4	Elemento 1,1,4	Elemento 2,0,4	Elemento 2,1,4

Fig. 6

Potete pensare ad una matrice tridimensionale come all'insieme delle pagine di un libro in cui ogni pagina contenga una tabella con lo stesso numero di righe e colonne. Nella figura sopra riportata (*Figura 6*) la nostra matrice è di tre pagine (per riprendere l'esempio del libro) numerate da 0 a 2 e ogni pagina contiene una tabella di 2 colonne e 5 righe, e nelle caselle di ogni elemento sono riportate le coordinate nella matrice.

Esempio di una matrice a due dimensioni

	A	B	C	D	E	F
1		**Anno**	**Anno1**	**Anno2**	**Anno3**	**Anno4**
2	**prova1**	500	410	320	250	150
3	**prova2**	520	440	330	280	180
4	**prova3**	545	425	335	275	175
5	**prova4**	595	485	300	205	105

Fig. 7

```
Sub demo2()
Dim i As Integer, n As Integer
Dim vendi(1 To 4, 1 To 5) As Long
vendi(1, 1) = 500
vendi(2, 1) = 520
vendi(3, 1) = 545
vendi(4, 1) = 595
vendi(1, 2) = 410
vendi(2, 2) = 440
vendi(3, 2) = 425
vendi(4, 2) = 485
vendi(1, 3) = 320
vendi(2, 3) = 330
vendi(3, 3) = 335
vendi(4, 3) = 300
vendi(1, 4) = 250
vendi(2, 4) = 280
vendi(3, 4) = 275
vendi(4, 4) = 205
vendi(1, 5) = 150
vendi(2, 5) = 180
vendi(3, 5) = 175
vendi(4, 5) = 105
 For i = 1 To 4
For n = 1 To 5
Foglio1.Cells(i + 1, n + 1) = vendi(i, n)
Next n
Next i
End Sub
```

Matrici Statiche e Dinamiche

In VBA possono essere create due tipi di matrici, matrici a dimensione fissa o statiche e matrici dinamiche. Una matrice che ha un numero fisso di elementi è una matrice di dimensioni fisse e viene utilizzato quando si conosce il numero preciso di elementi che verranno contenuti nella matrice, però la maggior parte delle volte sarà necessario creare array dinamico, perché non si sa l'esatta dimensione della matrice richiesta all'inizio e serve una certa flessibilità per modificare il numero di elementi della matrice.

Normalmente la dichiarazione di una matrice indica a VBA l'estensione delle varie dimensioni che provvede ad allocare una quantità di memoria sufficiente per tutti gli elementi della matrice, nel caso della matrice di *figura 1* VBA alloccherebbe memoria per 5 interi mentre per la matrice di figura 6 VBA alloccherebbe memoria per 10 elementi. VBA riserva spazio in memoria per tutti gli elementi della matrice finchè la relativa variabile esiste, matrici di questo tipo vengono dette **Statiche** perché il numero di elementi non cambia. Scegliere la dimensione di una matrice può essere complicato se non sapete a priori quanti dati dovrà ospitare o se la quantità di dati raccolti è molto variabile.

Se a volte gli elementi da memorizzare sono 100 e altre volte 10, potenzialmente vi troverete a sprecare lo spazio per memorizzare 90 elementi inutili (è la differenza tra il numero maggiore di elementi e il minore), per casi come questi il VBA prevede un tipo particolare di matrice, detto **dinamico**. Le matrici dinamiche sono definite in questo modo perché è possibile modificarne il numero di elementi durante l'esecuzione del programma e le matrici dinamiche se associate a una corretta programmazione possono crescere o stringersi per far posto esattamente al numero di elementi necessari eliminando lo spreco di spazio. Per modificare le dimensioni di una matrice dinamica si usa l'istruzione **Redim**.

E' possibile dichiarare una variabile dinamica con le dimensioni dell'indice vuoto e successivamente dimensionare o ridimensionare la matrice dinamica che è già stata dichiarata, utilizzando l'istruzione *ReDim*. Per ridimensionare un array, è necessario fornire il limite superiore, mentre il limite inferiore è facoltativo e se non si menziona verrà determinato dalle impostazione del modulo, che di default è *Option Base 0*. Per esempio è possibile dichiarare la matrice *prova1* come una matrice dinamica in questo modo: *Dim prova1 () As String*, per ridimensionare le dimensioni della matrice e portarla a 3 elementi (specificare Option Base 1), si utilizza l'istruzione Redim in questo modo: *ReDim prova1 (3) As String*, si può utilizzare una matrice dinamica invece di una a dimensione fissa, se si desidera regolare il numero di record nel database in fase di esecuzione.

```
Option Base 1
Sub prova2()

Dim prova1() As String
ReDim prova1(3) As String
prova1(1) = "Lunedì"
prova1(2) = "Martedì"
prova1(3) = "Mercoledì"
MsgBox prova1(1) & " - " & prova1(2) & " - " & prova1(3)
End Sub
```

Altro esempio di array dinamico

```
Option Base 1
Sub prova3()

Dim prova2() As String
ReDim prova2(3) As String
prova2(1) = "Lunedì"
prova2(2) = "Martedì"
prova2(3) = "Mercoledì"
ReDim prova2(4) As String
prova2(4) = "Giovedì"
MsgBox prova2(1) & " - " & prova2(2) & " - " & prova2(3) & " - " & prova2(4)
End Sub
```

Array dinamici – usare la parola chiave Preserve con l'istruzione ReDim

Quando un array viene ridimensionato utilizzando l'istruzione ReDim, i suoi valori si potrebbero perdere, per ovviare a questo possibile inconveniente e garantire che i valori della matrice non siano persi, si utilizza la parola chiave "*Preserve*" l'istruzione *ReDim*, che consentirà di conservare i dati esistenti nella matrice. Ad esempio, prima viene usata l'istruzione ReDim per dimensionare la matrice prova2: "*ReDim prova2 (3) As String*" che veniva popolata con 3 elementi, ora per ridimensionare la matrice e consentire la memorizzazione di 4 variabili senza perdere i dati esistenti, è necessario utilizzare l'istruzione "*ReDim Preserve myArray (4) As String*". Vedere l'esempio sotto riportato

```
Option Base 1
Sub demo4()
Dim New_mat() As String
ReDim New_mat(3) As String

New_mat(1) = "Primo"
New_mat(2) = "Secondo"
New_mat(3) = "Terzo"

ReDim Preserve New_mat(4) As String
New_mat(4) = "Quarto"
MsgBox New_mat(1) & " - " & New_mat(2) & " - " & New_mat(3) & " - " & New_mat(4)
End Sub
```

Si deve tenere presente che la dichiarazione Redim presenta le seguenti caratteristiche:

- Non può cambiare il tipo di dati della matrice
- Non può modificare il numero di dimensioni in un array
- Se si utilizza la parola chiave "*Preserve*", è possibile ridimensionare solo l'ultima dimensione della matrice, in modo che in un array multidimensionale gli stessi limiti devono essere specificati per tutte le altre dimensioni.

Vedi esempio sotto riportato. Si suppone di avere una tabella come mostrato nella figura sotto riportata che riporta le vendite per anno suddivise per reparto. Con le istruzioni che abbiamo appena visto si andrà a ridimensionare la matrice prima estendendola a 4 anni per poi ridurla a 2 anni.

	A	B	C	D	E	F
1		Anno	Anno1	Anno2	Anno3	Anno4
2	Reparto 1	500	410	320	250	150
3	Reparto 2	520	425	350	240	180
4	Reparto 3	545	485	390	290	190
5	Reparto 4	595	413	300	260	135

Fig. 8

```
Sub demo_6()
Dim annO As Integer, reP As Integer, ampliaM As Integer, limitI As Integer, limitS As Integer,
limitSR As Integer, rng As String

Dim matriC() As Integer
'la 1° dimensione specifica i 4 trimestri, la seconda dimensione specifica 2 anni di dati
'per i quali saranno inseriti i dati di vendita
ReDim matriC(1 To 4, 1 To 2)
'Stabilire i limiti superiori delle 2 dimensioni
limitI = UBound(matriC, 1)
limitS = UBound(matriC, 2)

For annO = 1 To limitS
For reP = 1 To limitI
matriC(reP, annO) = InputBox("Inserisci le vendite del reparto " & reP & " per l'anno " & annO)
Foglio1.Cells(reP + 1, annO + 1) = matriC(reP, annO)
Next reP
Next annO

If MsgBox("Continuare al prossimo anno?", vbQuestion + vbYesNo, "Confirmation") = vbYes Then
'con ReDim aumentiamo i dati di vendita a tre anni, dai 2 precedenti, per i 4 trimestri
ReDim matriC(1 To 4, 1 To limitS + 1)
'Determinare il limite superiore della seconda dimensione, dopo il ridimensionamento
limitSR = UBound(matriC, 2)
'si cicla per ciascuno dei 4 trimestri, inserendo i dati di vendita per l'anno aggiuntivo
For ampliaM = 1 To limitI

matriC(ampliaM, limitSR) = InputBox("Inserire i dati di vendita " & ampliaM & " per l'anno " &
limitSR)
Foglio1.Cells(1, 4) = "Anno 3"
Foglio1.Cells(ampliaM + 1, limitSR + 1) = matriC(ampliaM, limitSR)
Next ampliaM

End If
If MsgBox("Continuare con la riduzione della matrice?", vbQuestion + vbYesNo, "Confermare") =
vbYes Then
'è possibile inserire qualsiasi condizione, come cancellare i dati immessi in precedenza
With Sheets("Foglio1")
rng = .Name & "!" & .Cells(2, "B").Address & ":" & .Cells(limitI + 1, limitS + 2).Address
End With
Foglio1.Range(rng).ClearContents
'ora la matrice conterrà i dati di vendita per due anni
ReDim matriC(1 To 2, 1 To 2)
For annO = 1 To 2
For reP = 1 To 2
matriC(reP, annO) = InputBox("Inserisci i dati di vendita " & reP & " per l'anno " & annO)
Foglio1.Cells(reP + 1, annO + 1) = matriC(reP, annO)
Next reP
Next annO
End If
End Sub
```

Fig. 9

Terminata la fase di inserimento dei 2 anni, viene richiesta la conferma per continuare ai prossimi anni

Fig. 10

Inseriti i dati dell'anno richiesto ritorna un messaggio per ridurre la matrice

Fig. 11

Cliccando sul pulsante Si verranno cancellati i dati, ridotta la matrice e richiesti nuovi dati

Fig. 12

Una volta inseriti i dati richiesti la matrice presenta questo aspetto

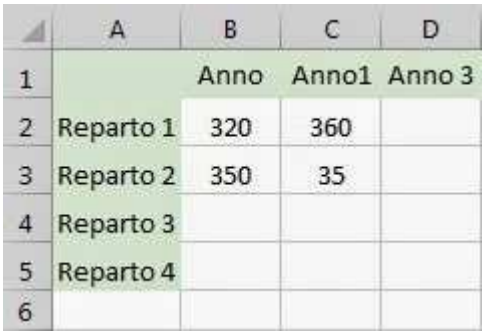


Fig. 13

Le Proprietà Range, Cells, Item, Row e Columns

È possibile fare riferimento o accedere a un Range del foglio di lavoro utilizzando le proprietà e i metodi dell'oggetto Range che si riferisce ad una cella o a un intervallo di celle, può essere una riga, una colonna o una selezione di celle comprendenti uno o più blocchi contigui di celle. Uno degli aspetti più importanti nella codifica vba fa riferimento a intervalli all'interno di un foglio di lavoro.

La Proprietà Range

Abbiamo detto che un oggetto ***Range*** si riferisce a una cella o a un intervallo di celle e può essere una riga, una colonna o una selezione di celle comprendenti uno o più blocchi contigui di celle. Un oggetto Range fa sempre riferimento a un foglio di lavoro specifico, e Excel attualmente non supporta gli oggetti Range che si sviluppano su più fogli di lavoro. Alcuni esempi di codice:

• Oggetto Range riferito a una singola cella
Dim rng As Range
Set rng = Range ("A1")

• Oggetto Range riferito a un blocco di celle contigue
Dim rng As Range
Set rng = Range ("A1: C3")

• Oggetto Range riferito ad una riga
Dim rng As Range
Set rng = Rows (1)

• Oggetto Range riferito a più colonne
Dim rng As Range
Set rng = Columns ("A: C")

Codice per l'oggetto Range riferito a 2 o più blocchi di celle contigue, utilizzando il metodo Union e Selection

Metodo Union

Codice per l'oggetto Range riferito a 2 o più blocchi di celle contigue, utilizzando il metodo Union e Selection

```
Dim rng1 As Range, rng2 As Range, rngUnion As Range
'impostare un blocco contiguo di celle come primo intervallo (o Range)
Set rng1 = Range ("A1: B2")
'impostare un altro blocco contiguo di celle come secondo intervallo
Set rng2 = Range ("D3: E4")
'assegnare una variabile (oggetto range) per rappresentare l'unione dei due intervalli
Set rngUnion = Union (rng1, rng2)
'colore interno impostato per l'intervallo che è l'unione di 2 oggetti Range
rngUnion.Interior.Color = vbYellow
```

Proprietà Selection

```
'Selezionare 2 blocchi contigui di celle, utilizzando il metodo Select
Range("A1:B2,D3:E4").Select
'impostare il colore di sfondo delle celle selezionate a giallo
Selection.Interior.Color = vbYellow
```

La proprietà Worksheet

Sintassi: *WorksheetObject.Range (Cell1, Cell2)*
E' possibile utilizzare solo l'argomento *Cell1* e in questo caso dovrà essere un riferimento a un Range che può includere un operatore di intervallo (2 punti) o l'operatore di unione (virgola), o il riferimento a un intervallo che può essere un nome definito. Esempi di utilizzo di questo tipo di riferimento sono:

Worksheets ("Foglio1"). Range ("A1") che si riferisce alla cella A1, oppure
Worksheets ("Foglio1"). Range ("A1: B3"), che si riferisce alle celle A1, A2, A3, B1, B2 e B3.

Quando entrambi gli argomenti cell1 e cell2 vengono utilizzati (cell1 e cell2 sono oggetti Range), si riferiscono alle celle comprese tra l'angolo superiore sinistro e l'angolo inferiore destro del Range, cioè le celle iniziali e finali del Range , e gli argomenti possono essere una singola cella, un'intera riga o colonna o una singola cella denominata. Un esempio di utilizzo di questo tipo di riferimento è

Worksheets ("Foglio1"). Range (Cells (1, 1), Cells (3, 2)), che si riferisce alle celle A1, A2, A3, B1, B2 e B3. Utilizzando il codice *Range ("A1")* verrà restituita la cella A1 del foglio attivo, come se si utilizza la sintassi *Application.Range ("A1")* o *ActiveSheet.Range ("A1")*

La proprietà Range

Sintassi: *RangeObject.Range (Cell1, Cell2)*
Per accedere a un Range relativo a un intervallo, ad esempio:
Worksheets("Foglio1").Range("C5:E8").Range("A1") si farà riferimento al Range ("C5") mentre il codice *Worksheets("Foglio1").Range("C5:E8").Range("B2")* farà riferimento al Range ("D6")

Rispetto ad utilizzare la proprietà Range, è possibile utilizzare anche un codice breve per fare riferimento a un intervallo utilizzando le parentesi quadre per racchiudere un riferimento di tipo "A1" o un nome. Durante l'utilizzo delle parentesi quadre, non si racchiude l'intervallo tra virgolette per renderlo una stringa. Utilizzando le parentesi quadre è come applicare il metodo *Evaluate* dell'oggetto Application, in cui la proprietà Range o il metodo Evaluate utilizzano un argomento stringa che permette di manipolare la stringa con codice VBA

Esempi: utilizzando

[A1]. Value = 5 è equivalente all'utilizzo di
Range ("A1"). Value = 5, mentre usando
[A1: A3, B2: B4, C3: D5]. Interior.Color = vbRed equivale a
Range ("A1: A3, B2: B4, C3: D5"). Interior.Color = vbRed e con intervalli denominati come:
[pippo] .Interior.Color = vbBlue equivale a
Range ("pippo"). Interior.Color = vbBlue.

Utilizzando le parentesi quadre si consentono solo riferimento a intervalli fissi, mentre invece usando la proprietà Range si permette di manipolare l'argomento stringa con codice VBA in modo che è possibile utilizzare delle variabili per fare riferimento a un Range dinamico, come illustrato di seguito:

```
Sub RangeDinamico ()
Dim i As Integer
'si inserisce Ciao nelle celle da 1 a 5 della colonna B
For i = 1 To 5
Range ("B" & i) = "Ciao"
next i
End Sub
```

La proprietà Cells

Restituisce un oggetto Range riferito a tutte le celle di un foglio di lavoro o a un intervallo, in quanto può essere utilizzata con riferimento ad un oggetto Application, a un oggetto foglio di lavoro o un oggetto Range. La proprietà ***Application.Cells*** si riferisce a tutte le celle del foglio di lavoro attivo ed è possibile utilizzare Application.Cells nel codice o omettere il qualificatore di oggetto e utilizzare il codice per riferirsi a tutte le celle del foglio di lavoro attivo.

La proprietà Worksheet.Cells

Sintassi: *WorksheetObject.Cells* si riferisce a tutte le celle di un foglio di lavoro specificato, utilizzando il codice: *Worksheets ("Foglio1"). Cells* si fa riferimento a tutte le celle del foglio denominato "Foglio1".

Utilizzando la struttura **Range.Cells** ci si riferisce alle celle in un intervallo specificato - Sintassi: *RangeObject.Cells*. Questa proprietà può essere utilizzata come *Range ("A1: B5"). Cells*, ma utilizzando le celle come nome intervallo in questo caso è irrilevante perché con o senza questa dicitura il codice farà riferimento al Range A1: B5. Per fare riferimento a una cella specifica, si deve utilizzare la **Proprietà Item** dell'oggetto Range, specificando la riga relativa e le posizioni delle colonne dopo la parola chiave Cells, vale a dire,

Worksheets ("Foglio1"). Cells.Item (2, 3) si riferisce al Range **C2** e
Worksheets ("Foglio1"). Range ("C2"). Cells (2, 3) farà riferimento al Range **E3.**

Poiché la proprietà **Item** è la proprietà predefinita del oggetto Range è possibile omettere questa istruzione utilizzando questo codice

Worksheets ("Foglio1"). Cells (2, 3), che si riferisce anche al Range C2.
Si può preferire in alcuni casi utilizzare
Worksheets ("Foglio1"). Cells (2, 3) rispetto a
Worksheets ("Foglio1"). Range ("C2")
perché le variabili di riga e colonna possono essere facilmente utilizzabili

La Proprietà Item: si deve utilizzare la proprietà ***Range.Item*** per restituire un intervallo come offset nell'intervallo specificato Sintassi: *RangeObject.Item (RowIndex, ColumnIndex)*. È necessario specificare l'argomento *RowIndex* mentre *ColumnIndex* è opzionale. RowIndex è il numero di indice della cella, partendo da 1 e crescente da sinistra a destra e poi verso il basso.

Worksheets ("Foglio1"). Cells.Item (1) o
Worksheets ("Foglio1"). Cells (1) si riferisce al Range A1, mentre
Worksheets ("Foglio1"). Cells (2) si riferisce al Range B1.

Durante l'utilizzo di un solo parametro di riferimento della proprietà Item (RowIndex), se l'indice supera il numero di colonne nell'intervallo specificato, il riferimento verrà disposto alle righe successive all'interno delle colonne del Range. Tralasciando l'oggetto qualificatore imposterà il foglio attivo. Cells (16385) che si riferisce al Range A2 del foglio attivo in Excel 2007 che ha 16384 colonne e cells (16386) si riferisce all'intervallo B2, e così via. Si noti inoltre che RowIndex e ColumnIndex sono offset e relativi nell'intervallo specificato (cioè rispetto all'angolo superiore sinistro del campo specificato). Entrambe le espressioni:

Range ("B3").Item(1) e
Range ("B3: D6").Item (1) si riferiscono al Range B3.

Il seguente codice si riferisce al Range D4:

Range ("B3: D6") Item (6) o
Range ("B3: D6"). Cells (6) o
Range ("B3: D6") (6). ColumnIndex si riferisce al numero di colonna della cella e può essere un numero che inizia con 1 o può essere una stringa che inizia con la lettera "A".

Worksheets ("Foglio1"). Cells (2, 3) e
Worksheets ("Foglio1"). Cells (2, "C") si riferiscono entrambi al Range C2 in cui RowIndex è 2 e ColumnIndex è 3 (colonna C),

Range ("C2"). Cells (2, 3) si riferisce al Range E3 nel foglio attivo, e
Range ("C2"). Cells (4, 5) si riferisce al Range G5 nel foglio attivo. Utilizzando:
Range ("C2"). Item (2, 3) e *Range ("C2"). Item (4, 5)* si ha lo stesso effetto e si riferiscono, rispettivamente, al Range E3:G5. Utilizzando:

Range ("C2: D3"). Cells (2, 3) e
Range ("C2: D3"). Cells (4, 5) sarà come riferirsi rispettivamente al Range E3 e al Range G5. Omettere la Voce Item esprimendo il codice in questo modo
Range ("C2: D3") (2, 3) e
Range ("C2: D3") (4, 5), ci si riferisce al Range E3 e al Range G5.

La Proprietà Columns nell'Oggetto Foglio di lavoro ha la seguente
Sintassi: *WorksheetObject.Columns* per riferirsi a tutte le colonne in un foglio di lavoro che vengono restituite come un oggetto Range. Esempio:

Worksheets ("Foglio1"). Columns restituirà tutte le colonne del foglio di lavoro, mentre
Worksheets ("Foglio1"). Columns (1), restituisce la prima colonna (colonna A) del foglio di lavoro, oppure *Worksheets ("Foglio1"). Columns ("A")* restituisce la prima colonna (colonna A) e
Worksheets ("Foglio1"). Columns ("A: C") restituisce le colonne A, B e C; e così via. Tralasciando l'oggetto qualificatore si imposterà il foglio come attivo, utilizzando la colonna (1) che restituisce la prima colonna del foglio attivo.

La Proprietà Columns dell'oggetto Range ha la seguente Sintassi: *RangeObject.Columns* e viene usato per fare riferimento alle colonne in un intervallo specificato. Esempio: per inserire un colore di sfondo nelle celle di tutte le colonne del Range specificato, cioè da B2 a D4.

Worksheets ("Foglio1"). Range ("B2: D4"). Columns.Interior.Color = vbYellow.
Oppure per Inserire il colore di fondo nelle celle della prima colonna del Range B2:B4:

Worksheets ("Foglio1"). Range ("B2: D4"). Columns (1). Interior.Color = vbGreen.

Se l'oggetto specificato contiene più zone, le colonne della prima area verranno restituite solo da questa proprietà. Prendiamo l'esempio di 2 aree nell'intervallo specificato, la prima area sarà "B2: D4" e la seconda "F3: G6", il seguente codice inserisce il colore di fondo nelle celle dalla prima colonna della prima area dalle celle B2: B4:

Worksheets ("Foglio1"). Range ("B2: D4, F3: G6"). Columns (1). Interior.Color = vbRed

La Proprietà Worksheet.Rows ha la seguente Sintassi: *WorksheetObject.Rows* e si riferisce a tutte le righe in un foglio di lavoro che vengono restituite come un oggetto Range si può usare

Worksheets ("Foglio1"). Rows che restituirà tutte le righe del foglio di lavoro, mentre
Worksheets ("Foglio1 "). Rows (1) restituisce la prima riga (riga uno) del foglio di lavoro e
Worksheets ("Foglio1"). Rows (3) restituisce la terza riga del foglio di lavoro e
Worksheets ("Foglio1"). Rows ("1:03") restituisce le prime 3 righe, e così via.

La Proprietà Range.Rows ha la seguente Sintassi: *RangeObject.Rows* fa riferimento a delle righe in un intervallo specificato, per esempio, per inserire il colore di fondo in tutte le righe specificate da B2 a D4 si usa:

Worksheets ("Foglio1"). Range ("B2: D4"). Rows.Interior.Color = vbYellow mentre invece per inserire il colore di fondo nella prima riga nel Range da B2 a D2:
Worksheets ("Foglio1"). Range ("B2: D4"). Rows (1). Interior.Color = vbGreen.

Se l'oggetto specificato contiene più zone, le righe della prima area verranno restituite solo dalla proprietà Areas. Prendiamo l'esempio di 2 aree nell'intervallo specificato, la prima area è "B2: D4" e la seconda area è "F3: G6" - il seguente codice inserisce il colore di fondo nelle celle dalla prima riga della prima area alle celle da B2 a D2:

Worksheets ("Foglio1"). Range ("B2: D4, F3: G6"). Rows (1). Interior.Color = vbRed.

La Proprietà Range.Offset ha la seguente Sintassi: *RangeObject.Offset (RowOffset, ColumnOffset)* . Entrambi gli argomenti sono facoltativi, l'argomento *RowOffset* specifica il numero di righe dell'intervallo specificato in cui si deve spostare, tenendo presente che valori negativi indicano uno spostamento verso l'alto e valori positivi indicano lo spostamento verso il basso, il valore di default è 0.
L'argomento *ColumnOffset* specifica il numero di colonne dell'intervallo specificato in cui ci si deve spostare, tenendo presente che valori negativi indicano lo spostamento a sinistra e valori positivi indicano lo spostamento a destra, il valore di default è 0. Esempio:

Range ("C5"). Offset (1, 2) si sposta di 1 riga e 2 colonne e si riferisce al Range E6, mentre invece
Range ("C5: D7 "). Offset (1, -2) si sposta di 1 riga verso il basso e di 2 colonne a sinistra e si riferisce al Range (A6: B8).

Accedere a un intervallo
con Riferimento a una singola cella, Inserire il valore 10 nella cella A1 del foglio di lavoro denominato "Foglio1"
Worksheets ("Foglio1"). Range ("A1"). Value = 10
Worksheets ("Foglio1"). Range ("A1") = 10

Inserire il valore di 10 nella cella C2 del foglio di lavoro attivo: *ActiveSheet.Cells (2, 3). Value = 10*

Riferimento a un intervallo di celle

Inserire il valore 10 nelle celle A1, A2, A3, B1, B2 e B3 del foglio attivo
ActiveSheet.Range. ("A1: B3"). Value = 10
ActiveSheet.Range ("A1", "B3"). Value = 10
ActiveSheet.Range (Cells (1, 1), Cells (3, 2)) = 10

Inserire il valore 10 nelle celle A1 e B3 del foglio denominato "Foglio1":
Worksheets ("Foglio1"). Range ("A1, B3"). Value = 10

Impostate il colore di sfondo (rosso) per le celle B2, B3, C2, C3, D2, D3 e H7 del foglio di lavoro denominato "Foglio3"
ActiveWorkbook.Worksheets ("Foglio3"). Range ("B2: D3, H7"). Interior.Color = vbRed

Inserire il valore 10 nell'intervallo denominato "pippo" del foglio di lavoro attivo, vale a dire che è possibile assegnare un nome al Range ("B2: B3") come "pippo" per inserire 10 nelle celle B2 e B3
Range ("pippo"). Value = 10
ActiveSheet.Range ("pippo"). Value = 10

Selezionare tutte le celle del foglio di lavoro attivo:
ActiveSheet.Cells.Select
Cells.Select

Impostare il font "Times New Roman" e la dimensione del carattere a 11, per tutte le celle del foglio di lavoro attivo nella cartella di lavoro attiva
ActiveWorkbook.ActiveSheet.Cells.Font.Name = "Times New Roman"
ActiveSheet.Cells.Font.Size = 11
Cells.Font.Size = 11

Righe e Colonne

Selezionare tutte le righe del foglio di lavoro attivo
ActiveSheet.Rows.Select

Inserire il valore 10 in ogni cella della riga 2 del foglio di lavoro denominato "Foglio1"
Worksheets ("Foglio1"). Righe (2). Valore = 10

Selezionare tutte le colonne del foglio di lavoro attivo
ActiveSheet.Columns.Select
Columns.Select

Inserire il valore 10 in ogni cella della colonna numero 3 del foglio di lavoro attivo
ActiveSheet.Columns (3). Value = 10
Columns ("C"). Value = 10

Inserire il valore 10 in ogni cella delle colonne 1, 2 e 3 del foglio denominato "Foglio1"
Worksheets ("Foglio1"). Columns ("A: C"). Value = 10

Riferimento relativo

Inserisce il valore 10 in C5
Range ("C5: E8"). Range ("A1") = 10

Inserisce il valore 10 in D6 - riferimento inizia dall'angolo superiore sinistro del campo definito:
Range ("C5: E8"). Range ("B2") = 10

Inserisce il valore 10 in E6, con offset di 1 riga e 2 colonne
Range ("C5"). Offset (1, 2) = 10

Inserisce il valore 10 nel Range ("F7: H10") con Offset di 2 righe e 3 colonne
Range. ("C5: E8"). Offset (2, 3) = 10

```
Sub esempio_1()
Dim ws As Worksheet, rng As Range
Dim r As Integer, c As Integer, n As Integer, i As Integer, j As Integer
Set ws = Worksheets("Foglio1")
ws.Activate
For r = 1 To 5
n = 1
For c = 1 To 5
Cells(r, c).Value = n
n = n + 1
Next c
Next r
Set rng = Range(Cells(1, 1), Cells(5, 5))
For i = 1 To 5
If i Mod 2 = 0 Then
rng.Columns(i).Interior.Color = vbYellow
Else
rng.Columns(i).Interior.Color = vbGreen
End If
Next i

For j = 1 To 5
If j Mod 2 = 0 Then
rng.Rows(j).Font.Bold = True
rng.Rows(j).Font.Color = vbRed
End If
Next j

rng.Cells.Font.Italic = True
End Sub
```

Per restituire il numero della prima riga in un intervallo, si utilizza la **Proprietà Range.Row** e se l'intervallo specificato contiene più aree, questa proprietà restituirà il numero della prima riga della prima area. Sintassi: *RangeObject.Row*, mentre invece per restituire il numero della prima colonna in un intervallo, si utilizza la **Proprietà Range.Column** e se l'intervallo specificato contiene più aree, questa proprietà restituirà il numero della prima colonna nella prima area. Sintassi: *RangeObject.Column*

Esempi:
Prendi il numero della prima riga nell'intervallo specificato – restituisce 4:
MsgBox ActiveSheet.Range ("B4"). Row
MsgBox Worksheets ("Foglio1"). Range ("B4: D7"). Row

Prendi il numero della prima colonna nell'intervallo specificato - restituisce 2:
MsgBox ActiveSheet.Range ("B4: D7"). Columns

Prendi il numero dell'ultima riga nell'intervallo specificato - restituisce 7:
Spiegazione: *Range ("B4: D7"). Rows.Count* restituisce 4 (il numero di righe nell'intervallo), così come: *Range("B4:D7").Rows(Range("B4:D7").Rows.Count)*o *Range("B4:D7").Rows(4)*, restituiscono l'ultima riga nell'intervallo specificato.

MsgBox Range ("B4: D7"). Row. (Range ("B4: D7"). Rows.Count). Row

Esempio: Utilizzare la Proprietà Row, Columns per determinare il numero di riga e numero di colonna a righe alternate

```
Sub esempio_2()
Dim rng As Range, cell As Range, i As Integer

Set rng = Worksheets("Foglio1").Range("B4:D7")

For Each cell In rng
cell.Value = cell.Row & "," & cell.Column
Next

For i = 1 To rng.Rows.Count
If i Mod 2 = 1 Then
rng.Rows(i).Interior.Color = vbGreen
Else
rng.Rows(i).Interior.Color = vbYellow
End If
Next
End Sub
```

È possibile ottenere un riferimento di intervallo in linguaggio VBA utilizzando la **proprietà Range.Address** , che restituisce l'indirizzo di un intervallo come valore stringa. Questa proprietà è di sola lettura. Esempi di utilizzo

MsgBox Range ("B2"). Address 'Restituisce $B$2

MsgBox Range ("B2, C3"). Address 'Restituisce $B$2, $C$3

Dim strRng As String 'Restituisce $A$1: $B$2, $C$3, $D$4
Range ("A1: B2, C3, D4"). Select
strRng = Selection.Address
MsgBox strRng

MsgBox Range ("B2"). Address (RowAbsolute: = False) 'Restituisce $B2

MsgBox Range ("B2"). Address (ColumnAbsolute: = False) 'Restituisce B$2

MsgBox Range ("B2"). Address (ReferenceStyle: = xlR1C1) 'Restituisce R2C2

Restituisce R[1] C[-1] – il Range ("B2") è di 1 riga e -1 colonna rispetto al Range ("C1")
MsgBox Range ("B2"). Address (RowAbsolute: = False, ColumnAbsolute: = False, ReferenceStyle: = xlR1C1, relativeTo: = Range ("C1"))

Restituisce RC [-2] – il Range ("A1") è di 0 riga e -2 colonne rispetto al Range ("C1")
MsgBox Cells(1, 1). Address (RowAbsolute: = False, ColumnAbsolute: = False, ReferenceStyle: = xlR1C1, relativeTo: = Range ("C1"))

Attivare e selezionare Celle con ActiveCell e Selection

Il metodo **Select** (dell'oggetto Range) viene utilizzato per selezionare una cella o un intervallo di celle in un foglio di lavoro e presenta la seguente Sintassi: *RangeObject.Select*, dovete assicurarvi che il foglio di lavoro in cui viene applicato il metodo Select per selezionare le celle, sia il foglio attivo.
La proprietà **ActiveCell** (dell'oggetto Application) restituisce una singola cella attiva (oggetto Range) nel foglio di lavoro attivo, tenendo presente che la proprietà ActiveCell non funziona se il foglio attivo non è un foglio di lavoro. Quando si seleziona una cella nella finestra attiva, la proprietà **Selection** (dell'oggetto Application) restituisce un oggetto Range che rappresenta tutte le celle che sono attualmente selezionate nel foglio di lavoro attivo.

Una selezione può essere costituita da una singola cella o un intervallo di più celle, ma ci sarà una sola cella attiva al suo interno, che viene restituita utilizzando la proprietà *ActiveCell*. Quando viene selezionata una sola cella, la proprietà ActiveCell restituisce questa cella, mentre invece selezionando più celle utilizzando il metodo *Select*, la prima cella di riferimento diventa la cella attiva, e, successivamente, è possibile modificare la cella attiva utilizzando il metodo Activate.

Sia la proprietà ActiveCell e la proprietà Selection sono di sola lettura, e non specificando l'oggetto Application cioè il qualificatore *Application.ActiveCell* o *ActiveCell* o *Application.Selection* o *Selection*, avranno lo stesso effetto. Per attivare una singola cella all'interno della selezione corrente, si deve utilizzare il metodo **Activate** (dell'oggetto Range) che presenta questa Sintassi: *RangeObject.Activate* , e la cella attivata verrà restituita utilizzando la proprietà ActiveCell.

Abbiamo discusso sopra che una selezione può essere costituita da una singola cella o un intervallo di più celle, mentre ci può essere solo una cella attiva all'interno della selezione e quando si attiva una cella al di fuori della selezione corrente, la cella attiva diventa l'unica cella selezionata. È inoltre possibile utilizzare il metodo Activate per specificare un intervallo di più celle, ma in effetti sarà attivata solo una singola cella, e questa cella attiva sarà la cella rappresentata dall'angolo superiore sinistro del campo specificato nel metodo. Se questa cella in alto a sinistra si trova all'interno della selezione, la selezione corrente non cambierà, ma se questa cella in alto a sinistra si trova al di fuori della selezione, l'intervallo specificato nel metodo *Activate* diventa la nuova selezione.

Vediamo del codice per illustrare i concetti di ActiveCell e Selection

Selezione contenente un intervallo di celle e la cella attiva
Range ("C1: F5"). Select 'selezionare il range C1:F5
MsgBox ActiveCell.Address 'restituisce C1, la prima cella di riferimento come cella attiva

Selezione contenente un intervallo di celle e la cella attiva
Range ("F5: C1"). Select 'selezionare il range F5:C1
MsgBox ActiveCell.Address 'restituisce C1, la prima cella di riferimento come cella attiva

Selezione contenente un intervallo di celle e la cella attiva:
Range ("C5: F1"). Select 'selezionare il Range C1: F5
MsgBox ActiveCell.Address 'restituisce C1, la prima cella di riferimento come la cella attiva

Attivare una cella all'interno della selezione corrente
Range ("B6: F10"). Select 'Selezionare il range B6: F10
MsgBox ActiveCell.Address 'restituisce B6, la prima cella di riferimento come la cella attiva
Range ("C8"). Activate 'La selezione è sempre la stessa, ma la cella attiva è ora C8
MsgBox ActiveCell.Address

Attivare una cella al di fuori della selezione corrente
Range ("B6: F10"). Select 'Seleziona l'intervallo B6: F10
MsgBox ActiveCell.Address 'restituisce B6, la prima cella di riferimento, come cella attiva
Range ("A2"). Activate 'la selezione e la cella attiva è ora A2
MsgBox ActiveCell.Address

Selezionare una cella all'interno della selezione corrente
Range ("B6: F10"). Select 'Selezionare l'intervallo B6: F10
MsgBox ActiveCell.Address 'restituisce B6, la prima cella di riferimento come la cella attiva
Range (" C8 "). Select 'sia la selezione che la cella attiva diventano C8
MsgBox ActiveCell.Address

Attivare un intervallo di celle in cui la cella in alto a sinistra si trova all'interno della selezione corrente
Range ("B6: F10"). Select 'Seleziona l'intervallo B6: F10
MsgBox ActiveCell.Address 'restituisce B6, la prima cella di riferimento, come cella attiva
Range ("C8: G12"). Activate 'La selezione rimane la stessa, ma la cella attiva è ora C8
MsgBox ActiveCell.Address

Attivare un intervallo di celle in cui la cella in alto a sinistra si trova al di fuori della selezione corrente
Range ("B6: F10"). Select 'Selezionare l'intervallo B6: F10
MsgBox ActiveCell.Address 'restituisce B6, la prima cella di riferimento, come la cella attiva
Range ("B1: F8"). Activate 'viene cambiato l'intervallo e la selezione, e la cella attiva è ora B1
MsgBox ActiveCell.Address

La proprietà Application.Selection restituisce l'oggetto selezionato in cui la selezione determina il tipo di oggetto restituito e quando la selezione è un intervallo di celle, questa proprietà restituisce un oggetto *Range*, e questa selezione (che è un oggetto Range), può comprendere una singola cella, o più celle o intervalli multipli non contigui. Come detto sopra, il metodo Select (dell'oggetto Range) è usato per selezionare una cella o un intervallo di celle in un foglio, pertanto, dopo aver selezionato un intervallo, è possibile eseguire azioni sulla selezione di celle utilizzando l'oggetto Selection.

```
Sub esempio()
'Selezionare le celle del foglio attivo utilizzando il metodo Range.Select
Range("A1:B3,D6").Select
'impostare il colore di sfondo rosso alle celle della selezione
Selection.Interior.Color = vbRed
End Sub
```

Le proprietà Entire Row, Entire Column e Insert

La proprietà **Range.EntireRow** restituisce un'intera riga o le righe all'interno del Range specificato e restituisce un oggetto Range con riferimento alla intera riga. Sintassi: *RangeObject.EntireRow*, mentre invece la proprietà **Range.EntireColumn** restituisce un'intera colonna o le colonne all'interno del Range specificato e restituisce un oggetto Range con riferimento alla intera colonna.
Sintassi: *RangeObject.EntireColumn*. Esempi di utilizzo delle proprietà EntireRow e EntireColumn

```
Range ("A2"). EntireRow.Select 'Selezionare la riga 2
Range ("A2: C4"). EntireRow.Select 'Selezionare le righe 2, 3 e 4
Cells (3, 4). EntireRow.Cells. (1, 1). Value = 3 'Inserire il valore 3 nel Range A3 esempio
Range ("A2"). EntireColumn.Select 'Selezionare la colonna A
Range ("A2: C4"). EntireColumn.Select 'Selezionare le colonne da A a C:
Cells (3, 4). EntireColumn.Cells (1, 1). Value = 4 'Inserire il valore 4 nel Range D1
```

Il metodo **Range.Insert** si utilizza per inserire una cella o un intervallo di celle in un foglio di lavoro.
Sintassi: *RangeObject.Insert (Shift, CopyOrigin)*. Entrambi gli argomenti racchiusi nelle parentesi sono facoltativi.
Quando si inseriscono delle celle, le altre vengono spostate per fare spazio a quelle inserite, ed è possibile impostare un valore per per determinare la direzione in cui le altre cellule devono spostarsi.
Specificando *xlShiftDown* si sposteranno le celle in basso e con *xlShiftToRight* si spostano le celle a destra.
Tralasciando questo argomento la direzione di spostamento verrà decisa in base alla forma del Range.
Specificando *xlFormatFromLeftOrAbove* per l'argomento *CopyOrigin* si copierà il formato delle celle inserite dalle celle sopra a sinistra, e specificando *xlFormatFromRightOrBelow* si copierà il formato dalle celle sotto a destra.

Spostare le celle in basso e copiare la formattazione della cella inserita dalla cella sopra della stessa colonna
Range ("B2"). Insert

Spostare le celle a destra e copiare la formattazione delle celle inserite dalla cella a sinistra
Range ("B2: C4"). Insert

Spostare le celle in basso e copiare la formattazione delle celle inserite a partire dalle celle di inserimento
Range ("B2: D3"). Insert

Spostare le celle in basso e copiare la formattazione delle celle inserite dalle celle in basso
Range ("B2: D3"). Insert CopyOrigin: = xlFormatFromRightOrBelow

Spostare le celle a destra e copiare la formattazione delle celle inserite dalle celle a
Range ("B2: D3"). Insert shift: = xlShiftToRight

Spostare le celle a destra e copiare la formattazione delle celle inserite dalle celle a destra
Range ("B2: D3"). Insert shift: = xlShiftToRight, CopyOrigin: = xlFormatFromRightOrBelow

Inserire 2 righe (la n° 2 e 3) e copiare la formattazione delle righe inserite dalle celle di sopra
Range ("B2: D3"). EntireRow.Insert

Di seguito sono riportati alcuni esempi di inserimento riga o colonna in modo dinamico in un foglio di lavoro.

Esempio: Inserire una riga, n righe sopra l'ultima riga utilizzata.

```
Sub inserisciR2()
Dim ws As Worksheet, rigaC As Long
Set ws = Worksheets("Foglio1")
rigaC = ws.Cells(Rows.Count, "A").End(xlUp).Row 'determinare l'ultima riga utilizzata nella colonna A
MsgBox rigaC
'si stabilisce il n° di righe da aggiungere sopra l'ultima riga utilizzata
n = 5
'se ci sono abbastanza righe prima dell'ultima riga utilizzato, altrimenti si otterrà un errore
If rigaC >= n Then
'se l'ultima riga utilizzata è 5 righe prima dell'inserimento, si inserisce come 1° riga
    ws.Rows(rigaC).Offset(-n + 1, 0).EntireRow.Insert
Else
    MsgBox "Non ci sono abbastanza righe prima dell'ultima riga utilizzata!"
End If
End Sub
```

Esempio: Inserire una riga ogni volta che il valore cercato viene trovato in un intervallo.

```
Sub inserisciR2()
    Dim ws As Worksheet, indirizzo1 As String
    Dim trovaR As Range, cercaR As Range, ultimaC As Range
Set ws = Worksheets("Foglio1")
Set cercaR = ws.Range("A1:K100") 'impostare intervallo di ricerca
    MsgBox "Ricerca di 'pippo' nel Range: " & cercaR.Address
Set ultimaC = cercaR.Cells(cercaR.Cells.Count) 'iniziare la ricerca dopo l'ultima cella nel range
'trovare il valore specificato, iniziando la ricerca dopo l'ultima cella nel range di ricerca
Set trovaR = cercaR.Find(What:="pippo", After:=ultimaC, LookIn:=xlValues, lookat:=xlWhole)

If trovaR Is Nothing Then
    MsgBox "Valore non trovato!"
    Exit Sub
Else
indirizzo1 = trovaR.Address
Do
    Set trovaR = cercaR.FindNext(After:=trovaR)
'quando viene trovato il valore, se il valore viene trovato due volte, verranno inserite 2 righe
    trovaR.Offset(1).EntireRow.Insert
Loop While trovaR.Address <> indirizzo1
End If
End Sub
```

Esempio: Inserire una riga quando un determinato valore viene trovato.

```
Sub inserisciR()
Dim ws As Worksheet
Dim cercaRNG As Range, cercaR As Range, ultimaRNG As Range
Dim ultimaR As Long

Set ws = Worksheets("Foglio1")
'si deve trovare un valore in questo range e dopo si inserisce una riga
Set cercaR = ws.Range("A1:E100")
'si inizia la ricerca dopo l'ultima cella del range di ricerca
Set ultimaRNG = cercaR.Cells(cercaR.Cells.Count)
Set cercaRNG = cercaR.Find(What:="pippo", After:=ultimaRNG, LookIn:=xlValues, lookat:=xlWhole)
If Not cercaRNG Is Nothing Then  'Procedura di uscita se il valore non viene trovato
ultimaR = cercaRNG.Row
MsgBox ultimaR
Else
MsgBox "Valore non trovato!"
Exit Sub
End If
'se il valore viene trovato (è nella riga 12), si inserisce una riga e passa alla n° 13
ws.Cells(ultimaR + 1, 1).EntireRow.Insert
' se il valore rilevato, si inserisce una riga 3 righe di seguito (riga n ° 15)
ws.Cells(ultimaR + 3, 1).EntireRow.Insert
'se il valore viene trovato, si inseriscono 3 righe e il valore alla riga 12 sarà spostato alla riga 15
ws.Cells(ultimaR, 1).Offset(3).EntireRow.Insert
'oppure
ws.Cells(ultimaR, 1).EntireRow.Resize(3).Insert
    ws.Range(Cells(ultimaR, 1), Cells(ultimaR + 2, 1)).EntireRow.Insert
    ws.Rows(ultimaR & ":" & ultimaR + 2).EntireRow.Insert Shift:=xlDown

    ws.Cells(ultimaR + 1, 1).EntireRow.Resize(3).Insert
    ws.Range(ws.Cells(ultimaR + 1, 1), ws.Cells(ultimaR + 3, 1)).EntireRow.Insert

ws.Cells(ultimaR + 2, 1).EntireRow.Resize(3).Insert
End Sub
```

Esempio: Inserire una riga o colonna, specificando la riga/colonna da inserire

```
Sub inserireRC()
    Dim ws As Worksheet
    Set ws = Worksheets("Foglio1")
'specificare il numero di righe da inserire
ws.Rows(12).Insert
'specificare il range sotto al quale inserire le righe
ws.Range("C3").EntireRow.Offset(1, 0).Insert
'specificare il numero di colonne da inserire
ws.Columns(4).Insert
'specificare l'intervallo a destra del quale inserire le colonne
ws.Range("C3").EntireColumn.Offset(0, 1).Insert
End Sub
```

Esempio: Inserisci righe (numero definito dall'utente) all'interno di valori consecutivi che si trovano in una colonna

```
Sub inserisciR3()
Dim ws As Worksheet, rng As Range
Dim ultimaRU As Long, rigaI As Long, Rcella As Long, Ccella As Long

Set ws = Worksheets("Foglio1")
ws.Activate
'impostare il numero di colonna in cui due valori consecutivi sono controllati per inserire le righe
Ccella = 1
'impostare il numero di riga da dove iniziare la ricerca dei valori consecutivi
Rcella = 1
'determinare l'ultima riga utilizzata nella colonna
ultimaRU = Cells(Rows.Count, Ccella).End(xlUp).Row
'inserire il numero di righe da inserire tra due valori consecutivi
rigaI = InputBox("Inserisci ilnumero di righe da inserire")

If rigaI < 1 Then
MsgBox "Errore - inserire un valore uguale o superiore a 1"
Exit Sub
End If

MsgBox "Questo codice inserirà " & rigaI & " righe, ovunque si trovano valori consecutivi nella
colonna numero " & Ccella & ", iniziando la ricerca dalla riga numero " & Rcella
'Loop fino al numero di riga corrisponde l'ultima riga utilizzata
Do While Rcella < ultimaRU
Set rng = Cells(Rcella, Ccella)
'Nel caso di due valori consecutivi
If rng <> "" And rng.Offset(1, 0) <> "" Then
'Inserire il numero di righe definito dall'utente
Range(rng.Offset(1, 0), rng.Offset(rigaI, 0)).EntireRow.Insert
Rcella = Rcella + rigaI + 1
'Determinare l'ultima riga utilizzata in modo dinamico e cambia l'inserimento delle righe
ultimaRU = Cells(Rows.Count, Ccella).End(xlUp).Row
'Metodo alternativo per determinare l'ultima riga utilizzata
Else
Rcella = Rcella + 1
End If
Loop

End Sub
```

Le righe in un foglio di lavoro

Sappiamo già che in un foglio di lavoro le informazioni vengono organizzate in colonne e righe e come per le colonne, ciascuna riga è etichettata e numerata. Queste etichette sono rappresentate da piccoli rettangoli posti sul lato sinistro dell'interfaccia di Excel e ogni rettangolo che mostra l'etichetta di una riga è chiamata intestazione di riga. E' possibile identificare le righe in un foglio di lavoro tramite la proprietà **Rows** di cui è dotato il foglio, pertanto, per fare riferimento a una riga, è possibile utilizzare l'oggetto foglio di lavoro per accedere alla proprietà Rows, in alternativa è possibile fare riferimento alle righe utilizzando l'oggetto **Range**.
Per identificare una riga si deve indicare il relativo foglio di lavoro e passare il suo indice tra le parentesi al metodo Rows, ecco un esempio che si riferisce alla 5 ° riga del secondo foglio di lavoro della cartella di lavoro corrente:

```
Sub Prova ()
    Workbooks.Item(1).Worksheets.Item(2).Rows(5)
End Sub
```

Come per le colonne, il codice funziona solo se il secondo foglio della cartella di lavoro corrente è attivo, se si esegue il codice mentre un foglio di lavoro diverso dal secondo è attivo, si riceverà un errore. Come già detto, è possibile fare riferimento a una riga utilizzando l'oggetto Range, passando una stringa in cui nelle parentesi, si inserisce il numero della riga, seguito da due punti, e seguito dallo stesso numero di riga. Ecco un esempio che si riferisce alla Riga 4.

```
Sub Prova ()
    Range("4:4")
End Sub
```

Se si vuole fare riferimento alle righe più di una volta, è possibile dichiarare una variabile di tipo Range e inizializzarla utilizzando l'operatore Set e assegnare l'intervallo che si desidera. Ecco un esempio:

```
Sub Prova ()
Dim Serie_r As Range
   Set Serie_r = Workbooks.Item(1).Worksheets.Item("Foglio1").Range("4:4")
Serie_r. Select
End Sub
```

Individuazione di un gruppo di righe

Un gruppo di righe viene definito un intervallo se sono una accanto all'altra e per fare riferimento alle righe in un intervallo, tra le parentesi della collezione Rows, si deve passare una stringa che corrisponda al numero della riga di una estremità, seguita da due punti, e seguito dal numero di riga dell'altra estremità. Ecco un esempio che si riferisce all'intervallo di righe da 2 a 6

```
Sub Prova ()
    Rows ("2:6")
End Sub
```

Le righe di un gruppo si qualificano come non adiacenti se non sono posizionate una accanto all'altra e per fare riferimento alle righe non adiacenti, si deve usare l'oggetto Range e usare una stringa da inserire nelle parentesi che rappresenta il numero di ogni riga seguita da due punti, e seguita dallo stesso numero di riga. Queste combinazioni sono separate da virgole. Ecco un esempio che si riferisce alle righe 3, 5 e 8

```
Sub Prova ()
    Range ("3:3, 5:5, 8:8")
End Sub
```

Selezione di una riga

E' possibile selezionare una riga o un gruppo di righe utilizzando il mouse, la tastiera, o una combinazione di entrambi. Per selezionare una riga con il mouse, si posiziona il cursore su un'intestazione di riga e si trasforma in una freccia rivolta verso destra, quindi basta clic sulla riga

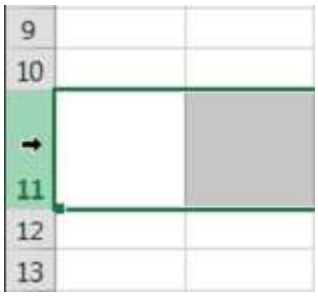


Fig. 1

È inoltre possibile utilizzare solo la tastiera per selezionare una riga, assicurandosi che una cella sul lato destro sia selezionata, premere e tenere premuto il tasto Shift e contemporaneamente premere la barra spaziatrice e rilasciare il tasto Maiusc. Per supportare la selezione la classe Rows è dotata di un metodo denominato Select, pertanto, per selezionare una riga utilizzando codice VBA, si deve accedere a una riga utilizzando i riferimenti che abbiamo visto in precedenza e richiamare il metodo Select. Ecco un esempio che seleziona la riga 6

```
Sub Prova ()
    Rows(6). Select
End Sub
```

Abbiamo anche visto che si potrebbe fare riferimento a una riga utilizzando l'oggetto Range, basta richiamare il metodo **Select** dopo l'accesso alla riga. Ecco un esempio che seleziona la riga 4

```
Sub Prova ()
    Range("4:4"). Select
End Sub
```

Selezione di un gruppo di righe

È anche possibile selezionare più di una riga sia utilizzando il mouse, la tastiera, o una combinazione di entrambi. Per selezionare un intervallo di righe con il mouse, si deve cliccare su una intestazione di riga e tenere premuto il mouse verso il basso (il cursore assume la forma di una croce bianca), quindi trascinare nella direzione del campo

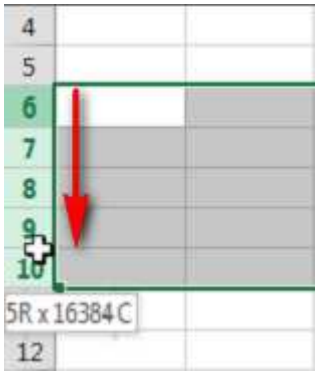


Fig. 2

Per selezionare più righe utilizzando solo la tastiera, si seleziona la riga di partenza e tenendo premuto il tasto Maiusc, premere il tasto freccia su o il tasto freccia giù. Una volta selezionato l'intervallo di righe, si rilascia il tasto Maiusc. È inoltre possibile utilizzare una combinazione di mouse e tastiera per selezionare una o più righe.

Per selezionare un intervallo di righe utilizzando una combinazione di mouse e tastiera, si sceglie una riga col mouse e si tiene premuto il tasto Shift, quindi si sceglie la riga all'altra estremità dell'intervallo e si rilascia il mouse. Per selezionare righe a caso utilizzando una combinazione di mouse e tastiera, si deve cliccare su una intestazione di riga, e tenendo premuto il tasto Ctrl si clicca su ogni intestazione di riga desiderata. Dopo aver selezionato le righe desiderate, si rilascia il mouse. Ogni riga selezionata sarà evidenziata

Fig. 3

Per selezionare un intervallo di righe utilizzando il codice, si deve fare riferimento al metodo Range, utilizzando le tecniche che abbiamo visto in precedenza, quindi richiamare il metodo **Select**. Ecco un esempio che seleziona le righe da 2 a 6

```
Sub Prova ()
    Rows("2:6"). Select
End Sub
```

Per selezionare righe non adiacenti, si deve fare riferimento alle tecniche viste in precedenza e richiamare il metodo Select. Ecco un esempio che seleziona le righe 3, 5 e 8

```
Sub Prova ()
    Range("3:3, 5:5, 8:8"). Select
End Sub
```

Per selezionare tutte le righe di un foglio di lavoro, si utilizza il metodo Select abbinandolo al metodo Rows. Ecco un esempio

```
Sub Prova ()
  Rows.Select
End Sub
```

Altezze delle righe

L'altezza di una riga è data dalla distanza della parte superiore dai bordi inferiori della riga e ci sono varie tecniche che è possibile utilizzare per modificare l'altezza di una riga. Per modificare manualmente l'altezza di una riga, si deve posizionare il mouse sul bordo inferiore che separa dalla riga successiva, il mouse si trasformerà come in *Fig. 4*, e poi si deve cliccare e trascinare verso l'alto o verso il basso fino a raggiungere l'altezza desiderata, quindi rilasciare il mouse.

Fig. 4

È inoltre possibile ridimensionare un gruppo di righe, in primo luogo, si devono selezionare le righe come abbiamo descritto in precedenza e poi posizionare il mouse sul bordo inferiore di una delle righe selezionate e cliccare e trascinare verso l'alto o verso il basso fino ad ottenere l'altezza desiderata. Quindi rilasciare il mouse.

Oppure si può modificare l'altezza di una riga cliccando sull'intestazione della riga e seguire il percorso dalla barra multifunzione **Home – Celle – Formato** e cliccare su Adatta Altezza righe

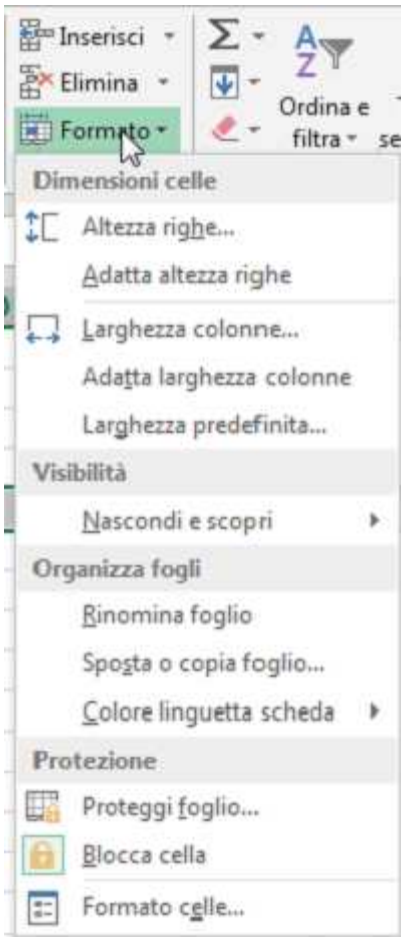


Fig. 5

È possibile utilizzare una finestra di dialogo per impostare esattamente all'altezza desiderata di una riga o di un gruppo di righe. Per specificare l'altezza di una riga si deve cliccare con il pulsante destro del mouse sull'intestazione della riga e nel menu che appare clic su Altezza righe. Per specificare la stessa altezza per molte righe si deve selezionare un intervallo di righe, come abbiamo visto in precedenza, e cliccare con il pulsante destro del mouse su una riga (o una delle intestazioni di riga o all'interno della selezione) e poi cliccare su Altezza righe nel menu a discesa che appare.

Per sostenere l'altezza di una riga, l'oggetto Row è dotato di una proprietà denominata **RowHeight**, pertanto, per specificare a livello di codice l'altezza di una riga, si deve accedere alla riga utilizzando un riferimento, come abbiamo visto in precedenza, e accedere alla sua proprietà RowHeight assegnando il valore desiderato. Ecco un esempio che imposta l'altezza della riga 6 a 2,50

```
Sub Prova ()
  Rows(6).RowHeight = 20
End Sub
```

Creazione di righe

Sappiamo che Excel crea e mette a disposizione più di un milione di righe (dalla versione 2007 in poi) che è possibile utilizzare quando si lavora su un foglio di lavoro e, a volte, può essere necessario inserire una riga tra due righe esistenti. Quando si aggiunge una nuova riga, Excel elimina l'ultima riga per mantenere il conteggio totale delle righe, e per inserire solo una nuova riga sopra uno esistente, si deve cliccare con il pulsante destro del mouse sull'intestazione della riga che sarà al di sotto della nuova che si desidera aggiungere e cliccare su Inserisci dal menu a discesa che compare, oppure cliccare sull'intestazione della riga o qualsiasi casella sul lato destro e dal percorso della barra multifunzione, **Home – Celle - Inserisci** e cliccare su Righe e dal menu a discesa che appare scegliere Inserisci righe foglio

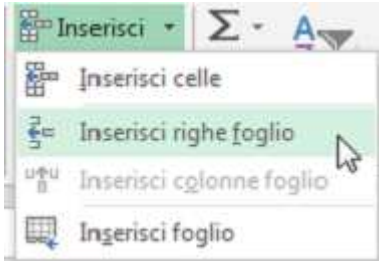


Fig. 6

E' possibile aggiungere una nuova riga usando il codice VBA, sfruttando la classe Rows che è dotata di un metodo denominato **Insert**, pertanto, per aggiungere una riga di codice, si deve fare riferimento alla riga che si posizionerà al di sotto della nuova e chiamare il metodo Insert. Ecco un esempio:

```
Sub Prova ()
    Rows(3). Insert
End Sub
```

Per aggiungere più di una riga, sia adiacenti che non adiacenti, si devono seguire i metodi poco sopra esposti, ricordando che la nuova si verrebbe a creare al di sotto delle righe selezionate. Per aggiungere nuove righe a livello di programmazione, si deve fare riferimento alle righe interessate e richiamare il metodo Insert. Ecco un esempio che aggiungerà nuove righe nelle posizioni 3, 6, e 10

```
Sub Prova ()
  Range("3:3, 6:6, 10:10").Insert
End Sub
```

Eliminare una riga

Per eliminare una riga si deve cliccare con il pulsante destro del mouse sull'intestazione della riga e fare clic su Elimina nel menu a discesa che appare, oppure dalla barra multifunzione, seguendo il percorso **Home – Celle - Elimina** quindi dal menu a discesa che appare cliccare su Elimina righe foglio

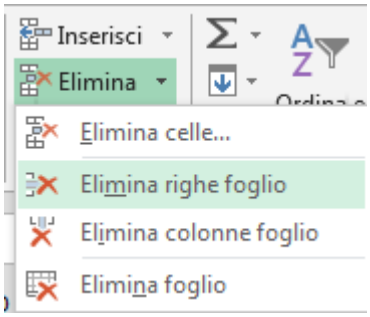


Fig. 7

Per supportare la rimozione riga via codice, la classe Row è dotata di un metodo chiamato **Delete** che non necessita di alcun argomento per individuare la riga. Sulla base di questo, per eliminare una riga, si deve accedere utilizzando un riferimento come abbiamo visto prima, e richiamare il metodo Delete. Ecco un esempio che cancella la riga 3

```
Sub Prova ()
  Rows(3).Delete
End Sub
```

Naturalmente, è possibile utilizzare anche l'oggetto Range per fare riferimento alla riga. Per cancellare più di una riga, prima si devono selezionare e seguire i percorsi sopra esposti, mentre per eliminare un gruppo di righe via codice si utilizza l'0ggetto Range in questo modo

```
Sub Prova ()
    Range("3:3, 6:6, 10:10").Delete
End Sub
```

Per nascondere una riga si deve selezionare la riga e fare clic su Nascondi dal menu a discesa che compare, oppure seguire il percorso dalla barra multifunzione **Home – Celle - Formato**, quindi cliccare sulla voce Nascondi e Scopri e dal menu a discesa che appare cliccare su Nascondi righe

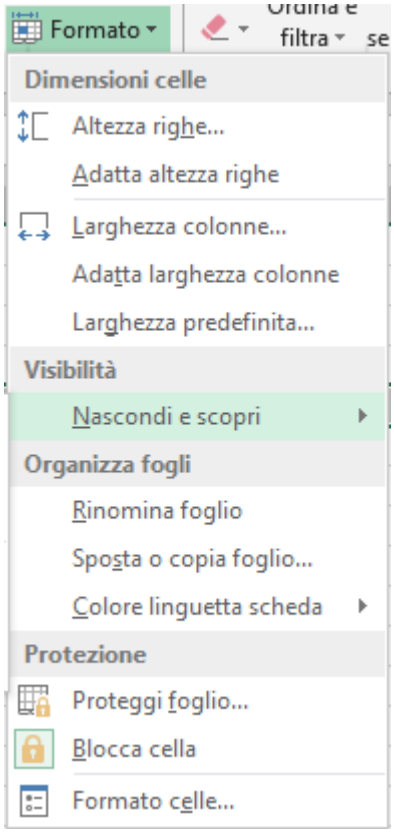


Fig. 8

Per nascondere una riga via codice, si deve prima selezionare, quindi usare la proprietà **EntireRow**. Ecco un esempio che nasconde la riga 6

```
Private Sub Prova ()
    Rows("6:6").Select
    Selection.EntireRow.Hidden = True
End Sub
```

Allo stesso modo, per nascondere un gruppo di righe, prima si deve selezionare il Range, quindi usare il comando Selection.EntireRow.Hidden = True, mentre per mostrare le righe nascoste basta cambiare il valore assegnato da TRUE a FALSE

Le Selezioni condizionali

Se si dispone di un gran numero di condizioni da esaminare, la dichiarazione *If ... Then ... Else* è costretta a verificarle tutte, ma il linguaggio Visual Basic offre l'alternativa di saltare direttamente all'istruzione che si applica allo stato di una condizione. Questo processo è indicato come selezionare le condizioni utilizzando le parole chiave **Select Case** e **Case**, la cui formula è:

```
Select Case Expression
   Case Expression1
        Statement1
   Case Expression2
      Statement2
   Case ExpressionX
      StatementX
End Select
```

La dichiarazione inizia con **Select Case** e termina con **End Select** e sul lato destro del Select Case, si deve immettere il valore, *Expression*, che verrà utilizzato, che può essere un valore booleano, un carattere o una stringa, un numero naturale, un numero decimale, un valore di data o ora, oppure una enumerazione.

All'interno del *Select Case* e prima della linea *End Select*, è necessario fornire una o più sezioni e ognuna contiene una parola chiave **Case** seguita da un valore sul lato destro e i valori di *Expresion1*, *Expresion2* o *ExpresionX*, devono essere dello stesso tipo come il valore di Expression. Dopo il Case e la sua espressione (Expression), si può scrivere una dichiarazione e quando si accede a questa sezione di codice, viene considerato il valore di Expression e viene confrontato con ciascun valore di Expression di ogni Case:

- Se il valore di Expression1 è uguale a quella di Expression, allora statement1 viene eseguito.
- Se il valore di Expression1 non è uguale a quella di Expression, l'interprete si sposta Expression2
- Se il valore di Expression2 è uguale a quella di Expression, allora viene eseguita l'istruzione Statement2
- Questo continuerà fino all'ultimo ExpressionX

Ecco un esempio:

```
Sub Test()
   Dim risp As Byte

   risp = CByte(InputBox( _
              "Una delle seguenti parole non è una parola chiave di Visual Basic" & vbCrLf &
vbCrLf & _
                "1) Function" & vbCrLf & _
                "2) Except" & vbCrLf & _
                "3) ByRef" & vbCrLf & _
                "4) Each" & vbCrLf & vbCrLf & _
                "Inserisci la tua risposta "))

   Select Case risp
      Case 1
         MsgBox ("Giusto: Function è una parola chiave di Visual Basic " & vbCrLf & _
               "Viene utilizzata per creare una procedura di tipo funzione")
      Case 2
         MsgBox ("Sbagliato: Except non è una parola chiave di " & vbCrLf & _
               "Visual Basic ma è una parola chiave " & vbCrLf & _
               "utilizzata in C++ per la Gestione delle eccezioni")
      Case 3
         MsgBox ("Giusto: ByRef è una parola chiave di Visual Basic" & vbCrLf & _
               "Viene utilizzata per passare un argomento a una procedura")
      Case 4
         MsgBox ("Giusto: Each è una parola chiave di Visual Basic " & vbCrLf & _
               "utilizzata in un ciclo per scorrere una lista di voci ")
   End Select
End Sub
```

Ecco un esempio di esecuzione del programma:

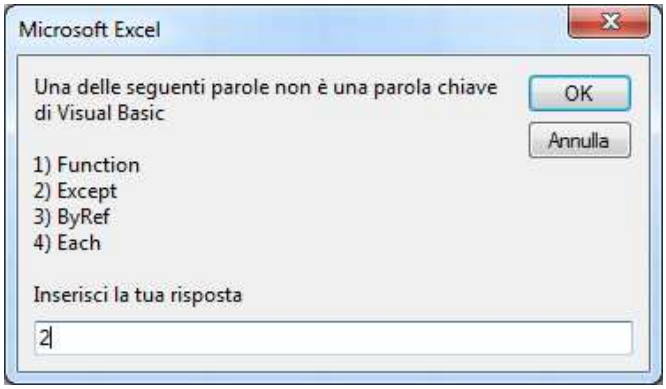


Fig. 1

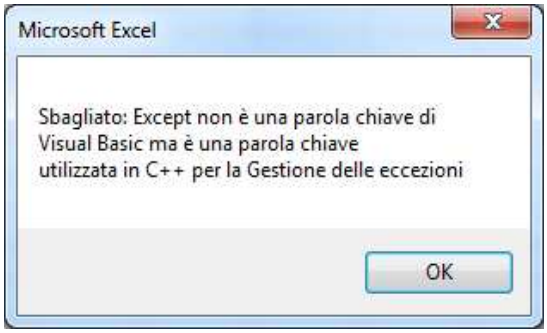


Fig. 2

Il codice di cui sopra presuppone che uno dei Case corrisponderà al valore di Expression, ma non è sempre così. Se si prevede che ci potrebbe essere nessuna corrispondenza tra l'espressione (Expression) e uno dei vari Expression dei Case, è possibile utilizzare una dichiarazione Case Else alla fine della lista. La dichiarazione sarebbe allora simile a questa:

```
Select Case Expression
   Case Expression1
      Statement1
   Case Expression2
      Statement2
   Case Expression3
      Statement3
   Case Else
      Statementk
End Select
```

In questo caso, la dichiarazione dopo *Case Else* viene eseguita se nessuna delle espressioni precedenti corrisponde alla espressione principale. Ecco un esempio:

```
Sub Test()
   Dim risp As Byte

   risp = CByte(InputBox( _
              "Una delle seguenti parole non è una parola chiave di Visual Basic" & vbCrLf &
vbCrLf & _
                "1) Function" & vbCrLf & _
                "2) Except" & vbCrLf & _
                "3) ByRef" & vbCrLf & _
                "4) Each" & vbCrLf & vbCrLf & _
                "Inserisci la tua risposta "))

   Select Case risp
      Case 1
         MsgBox ("Giusto: Function è una parola chiave di Visual Basic " & vbCrLf & _
               "Viene utilizzata per creare una procedura di tipo funzione")
      Case 2
         MsgBox ("Sbagliato: Except non è una parola chiave di " & vbCrLf & _
               "Visual Basic ma è una parola chiave " & vbCrLf & _
               "utilizzata in C++ per la Gestione delle eccezioni")
      Case 3
         MsgBox ("Giusto: ByRef è una parola chiave di Visual Basic" & vbCrLf & _
               "Viene utilizzata per passare un argomento a una procedura")
      Case 4
         MsgBox ("Giusto: Each è una parola chiave di Visual Basic " & vbCrLf & _
               "utilizzata in un ciclo per scorrere una lista di voci ")
      Case Else
         MsgBox ("Selezione non valida")
   End Select
End Sub
```

Ecco un esempio di esecuzione del programma:

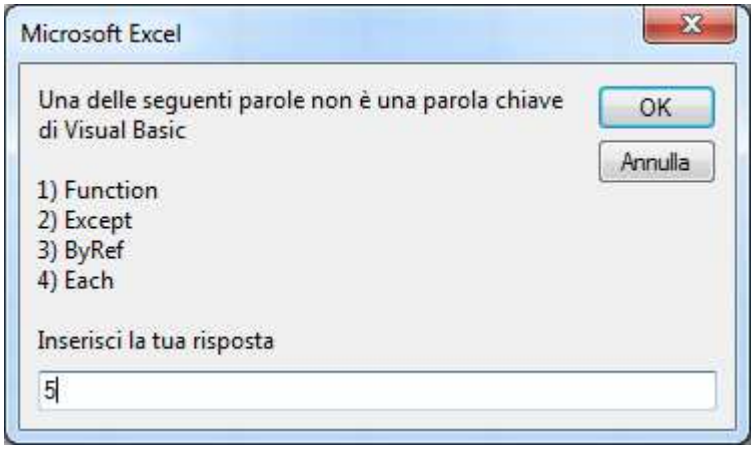


Fig. 3

Fig. 4

Come accennato nell'introduzione, il Select Case può utilizzare un valore diverso da un numero intero, ad esempio, è possibile utilizzare un carattere:

```
Sub Test()
    Dim sesso As String
    sesso = "M"

    Select Case sesso
        Case "F"
            MsgBox ("Femmina")
        Case "M"
            MsgBox ("Maschio")
        Case Else
            MsgBox ("Sconosciuto")
    End Select
    End Sub
```

Questo produrrebbe:

Fig. 5

Si noti che in questo caso stiamo usando solo caratteri maiuscoli, se si desidera convalidare anche caratteri minuscoli, potremmo essere costretti a creare sezioni aggiuntive di Case per ciascuno. Ecco un esempio:

```
Sub Test()
    Dim sesso As String
    sesso = "f"

    Select Case sesso
        Case "F"
            MsgBox ("Femmina")
        Case "f"
            MsgBox ("Femmina")
        Case "M"
            MsgBox ("Maschio")
        Case "m"
            MsgBox ("Maschio")
        Case Else
            MsgBox ("Sconosciuto")
    End Select
    End Sub
```

Questo produrrebbe:

Fig. 6

Invece di utilizzare un valore per Case, è possibile applicare più di una condizione, per fare ciò, sul lato destro della parola chiave Case, è possibile separare le espressioni con una virgola. Ecco alcuni esempi:

```
Sub Test()
    Dim sesso As String
    sesso = "F"

    Select Case sesso
        Case "f", "F"
            MsgBox ("Femmina")
        Case "m", "M"
            MsgBox ("Maschio")
        Case Else
            MsgBox ("Sconosciuto")
    End Select
    End Sub
```

Convalida di una serie di casi

È possibile utilizzare un intervallo di valori per un Case, a tale scopo, sul lato destro del Case, si deve inserire il valore più basso, seguito dalla parola chiave To e seguito dal valore più alto. Ecco un esempio:

```
Sub Test()
    Dim eta As Integer
    eta = 24
    Select Case eta
        Case 0 To 17
            MsgBox ("Ragazzo")
        Case 18 To 55
            MsgBox ("Adulto")
        Case Else
            MsgBox ("Anziano")
    End Select
End Sub
```

Questo produrrebbe:

Fig. 7

Verificare un valore

Si consideri la seguente procedura:

```
Sub Test()
    Dim num As Integer
    num = 448

    Select Case num
        Case -602
            MsgBox ("-602")
        Case 24
            MsgBox ("24")
        Case 0
            MsgBox ("0")
    End Select
End Sub
```

Ovviamente questa Select Case lavorerà in rari casi solo quando l'espressione di un Case corrisponde esattamente al valore richiesto, in realtà, per questo tipo di scenario, è possibile convalidare un intervallo di valori.

Il linguaggio Visual Basic fornisce un'alternativa, infatti è possibile controllare se il valore della espressione risponde a un criterio anziché a un valore esatto. Per crearla, si utilizza l'operatore **Is** con la seguente formula:

Is Operator Value

Si inizia con la parola chiave Is seguita da uno degli operatori booleani che abbiamo visto precedentemente, come: =, <>, <, <=,> o> = e sul lato destro dell'operatore booleano, si digita il valore desiderato. Ecco alcuni esempi:

```
Sub Test()
   Dim num As Integer
   num = -448

   Select Case num
      Case Is < 0
         MsgBox ("Il numero è negativo")
      Case Is > 0
         MsgBox ("Il numero è positivo")
      Case Else
         MsgBox ("Il numero è 0")
   End Select
End Sub
```

Anche se in questo esempio abbiamo usato un numero naturale, è possibile utilizzare qualsiasi confronto logico appropriato in grado di produrre un risultato vero o un falso, inoltre è possibile combinare con le altre alternative che abbiamo visto in precedenza, come ad esempio separando le espressioni di un Case con una virgola. Con la dichiarazione Select Case, abbiamo visto come controllare valori diversi e di intervenire. Ecco un esempio:

```
Sub Test()
   Dim num As Integer, cat As String
   num = 2

   Select Case num
      Case 1
         cat = "Ragazzo"
      Case 2
         cat = "Adulto"
      Case Else
         cat = "Anziano"
   End Select

   MsgBox ("La categoria è : " & cat)
End Sub
```

Questo produrrebbe:

Fig. 8

Abbiamo anche visto che il linguaggio Visual Basic fornisce la funzione **Choose** che può controllare una condizione e intraprendere un'azione, inoltre è un'altra alternativa a una dichiarazione Select Case. Se prendiamo in considerazione la sintassi della funzione Choose:

```
Function Choose( ByVal Index As Double,  ByVal ParamArray Choice() As Variant ) As Object
```

Si può notare che questa funzione richiede due argomenti, il primo argomento è equivalente a *Expression* della nostra Select Case, e come già detto, il primo argomento deve essere un numero che sarà il valore centrale rispetto al quale saranno confrontati gli altri valori. Invece di utilizzare sezioni Case si forniscono i corrispondenti valori di *ExpressionX* come un elenco di valori al posto del secondo argomento, separandoli da virgole. Ecco un esempio:

```
Choose(num, "Ragazzo", "Adulto", "Anziano")
```

Come già menzionato, i valori del secondo argomento sono forniti come una lista e ogni membro della lista utilizza un indice. Il primo membro della lista, che è il secondo argomento di questa funzione, ha un indice pari a 1, il secondo valore dell'argomento, che è il terzo argomento della funzione, ha un indice di 2. È possibile continuare ad aggiungere i valori del secondo argomento come meglio si crede. Quando la funzione Choose è stata richiamata, restituisce un valore di tipo Variant ed è possibile recuperare tale valore, memorizzandolo in una variabile e usarlo quando se ne ha la necessità. Ecco un esempio:

```
Sub Test()
    Dim num As Integer, cat As String
    num = 1
    cat = Choose(num, "Ragazzo", "Adulto", "Anziano")
    MsgBox ("Il tipo di categoria è: " & cat)
End Sub
```

Questo produrrebbe:

Fig. 9

Fino ad ora, abbiamo visto come creare istruzioni condizionali normali. Ecco un esempio:

```
Sub Test()
   Dim num%
   num% = InputBox("Inserisci un numero inferiore a 5")

   If num% >= 0 Then
      MsgBox (num%)
   End If
End Sub
```

Quando questa procedura viene eseguita, l'utente è invitato a fornire un numero, se questo numero è positivo, verrà visualizzata una finestra di messaggio, mentre se l'utente inserisce un numero negativo, non succede nulla. In un tipico programma, dopo la convalida di una condizione, si consiglia di agire e per fare questo, è possibile creare una sezione del programma all'interno della istruzione condizionale di convalida, ma in realtà, è possibile creare un'istruzione condizionale all'interno di un'altra istruzione condizionale. Questo è indicato come nidificazione condizione. Qualsiasi condizione può essere nidificato all'interno di un'altra e possono essere incluse più condizioni all'interno di un'altra. Ecco un esempio in cui una condizione If nidificata all'interno di un altro If:

```
Sub Test()
   Dim num%
   num% = InputBox("Inserisci un numero inferiore a 5")

   If num% >= 0 Then
      If num% < 12 Then
      MsgBox (num%)
   End If
   End If
End Sub
```

L'istruzione Goto

L'istruzione **Goto** consente di passare ad un'altra sezione del programma mentre è in esecuzione e per poter utilizzare l'istruzione Goto, si deve inserire un nome su una particolare sezione del procedimento in modo da poter fare riferimento a tale nome. Il nome, chiamato anche etichetta, è fatto di una sola parola e segue le regole che abbiamo applicato ai nomi e poi seguito da due punti ":". Ecco un esempio:

```
Sub Test()
Etichetta1:
End Sub
```

Dopo aver creato l'etichetta, è possibile elaborarla, inserendo nel codice l'istruzione GoTo seguita dall'etichetta, in modo tale che se si verifica una condizione che richiede di inviare il flusso all'etichetta corrispondente dopo la parola chiave Goto. Ecco un esempio:

```
Sub Test
   Dim num%
   num% = InputBox("Inserisci un numero inferiore a 5")

   If num% < 0 Then
      GoTo Nnum
   Else
      MsgBox (num%)
   End If

Nnum:
   MsgBox "Hai inserito un numero negativo"
End Sub
```

Allo stesso modo, è possibile creare un numero di etichette necessarie e fare riferimento ad esse quando si vuole. Il nome deve essere unico nel suo campo di applicazione e ciò significa che ogni etichetta deve avere un nome univoco nella stessa procedura. Ecco un esempio con varie etichette:

```
Sub Test()
   Dim risp As Byte

   risp = InputBox(" **__ Voci Multiple __** " & vbCrLf & vbCrLf & _
                   "Per creare una costante nel codice, " & _
                   "devi utilizzare la parola chiave Constant" & vbCrLf & _
                   "Cosa scegli (1=True/2=False)? ")
   If risp = 1 Then GoTo sb
   If risp = 2 Then GoTo gt

sb:
   MsgBox ("La parola chiave utilizzata per creare una costante è Const")
   GoTo Lv
gt:  MsgBox ("Constant non è una parola chiave")
Lv:
End Sub
```

Ecco un esempio di esecuzione del programma con risposta = 1:

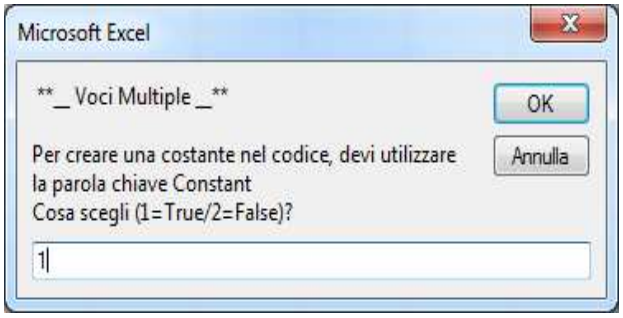


Fig. 10

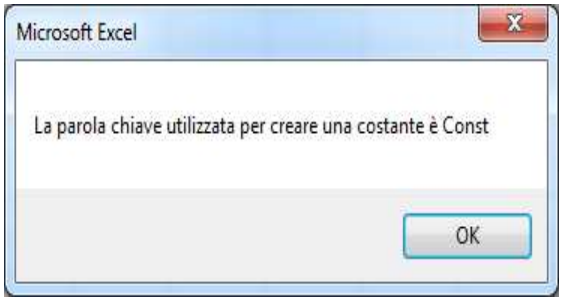


Fig. 11

Ecco un altro esempio di esecuzione dello stesso programma con risposta = 2:

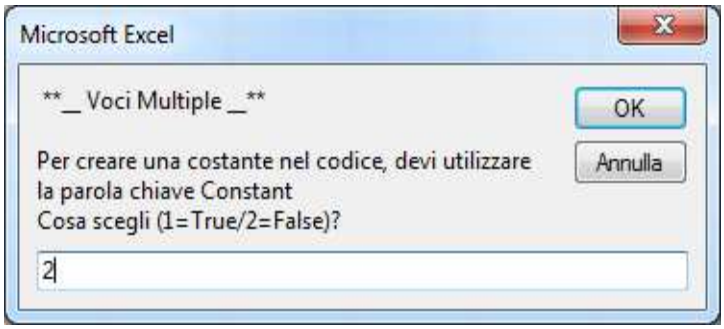


Fig. 12

Fig. 13

La negazione di una istruzione condizionale

Fino ad ora, abbiamo visto come un'istruzione condizionale che è vera o falsa, ma è possibile invertire il valore di una condizione, rendendolo Falso (o Vero). A sostegno di questa operazione, il linguaggio Visual Basic fornisce un operatore chiamato **Not**. La sua formula è:

Not Expression

Quando si scrive l'istruzione **Not**, deve essere seguita da un'espressione logica che può essere una semplice espressione booleana. Ecco un esempio:

```
Sub Test()
   Dim sposata As Boolean

   MsgBox ("La Signora è : " & sposata)
   MsgBox ("La Signora è : " & Not sposata)
End Sub
```

In questo caso, l'operatore Not viene utilizzato per modificare il valore logico della variabile e quando una variabile booleana è stata "negata", il suo valore logico è cambiato. Se il valore logico fosse stato Vero , sarebbe cambiato in False e viceversa, pertanto, è possibile invertire il valore logico di una variabile booleana di "Notting" o no "Notting" esso. Consideriamo ora il seguente programma:

```
Sub Test()
   Dim dsp As Boolean, tax As Double
   tax = 33#
   MsgBox ("Importo tassato : " & tax & "%")
   dsp = True
   If dsp = True Then
      tax = 30.65

      MsgBox ("Importo tassato : " & tax & "%")
   End If
End Sub
```

Questo produrrebbe:

Fig. 14

Fig. 15

Probabilmente il modo più classico di utilizzare l'operatore **Not** consiste in una "retromarcia" di un'espressione logica e per fare questo, si precedere l'espressione logica con l'operatore Not. Ecco un esempio:

```
Sub Test()
    Dim dsp As Boolean, tax As Double
    tax = 33#
    MsgBox ("Importo tassato : " & tax & "%")
    dsp = True

    If Not dsp Then
        tax = 30.65

        MsgBox ("Importo tassato : " & tax & "%")
    End If
End Sub
```

Questo produrrebbe:

Fig. 16

Allo stesso modo, è possibile negare qualsiasi espressione logica.

Leggere un File Txt con VBA

Ci sono vari modi di leggere un file Txt da VBA, vediamo un modo che credo sia facile da capire e modificare ma che richiede la conoscenza di un paio di concetti di VBA come la progettazione delle macro e la scelta dei metodi. Per meglio comprendere, vediamo quando, perché e come scegliere la funzione giusta, e una corretta gestione degli errori e delle decisioni da intraprendere a livello codice utilizzando una tecnica e metodi differenti. Cominciamo con l'aggiunta dei riferimenti necessari al progetto VBA, aprite un nuovo file Excel a dall'editor di VB seguite il percorso dal menu **Strumenti – Riferimenti**, scorrete la lista fino a trovare la libreria *Microsoft Scripting Runtime library* e spuntare la casella e confermate l'operazione cliccando sul tasto Ok. La Microsoft Scripting Runtime library ci permette di utilizzare:

FileSystemObject: Con questo oggetto si può accedere rapidamente a qualsiasi unità, cartella o file.
TextStream: E' un oggetto usato per leggere il contenuto di un file Txt

Creiamo ora un file Txt aprendo notepad o un altro editor di testo e copiate e incollate i dati di esempio qui sotto

```
Questa è la riga uno
questa invece è la seconda linea
è questa è la linea n ° 3
ecco la quarta riga del file
e finiamo con la Linea 5
```

Salvate il file col nome di *prova.txt* sul desktop e ritornate nell'editor di VBE per inserire un nuovo modulo, seguendo il percorso dal menu **Inserisci – Modulo** e copiate e incollate il codice sotto riportato

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream
   Set alex_1 = alex.OpenTextFile("C:\Test\prova.txt")
    alex_1.Close
End Sub
```

Ora se si esegue la procedura *leggi_txt* si ottiene un messaggio di errore del tipo

Fig. 1

E cliccando sul pulsante *Debug* si evidenzia la riga che ha rimandato l'errore

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream
   Set alex_1 = alex.OpenTextFile("C:\Test\prova.txt")
    alex_1.Close
End Sub
```

Fig. 2

L'errore rimandato è: *Impossibile trovare il file*. Un occhio attento si sarebbe accorto dell'errore prima di eseguire la procedura in quanto è evidente la differenza dei percorsi del file, ma è stato simulato questo errore per introdurre un modo per gestire questo errore. All'inizio è stato citato il metodo come semplice e ora vediamo di applicare questa logica.
Come si può vedere il metodo *OpenTextFile ()* accetta un parametro che è il percorso del file, ma la procedura non controlla se il file esiste o no, tocca al programmatore fornire un corretto percorso di ingresso alla funzione per evitare errori *Runtime*. Per essere sicuri che il file esista si possono usare vari metodi, ma il metodo più semplice sarebbe di pilotare il codice in modo che facesse questa valutazione: *se il file esiste puoi proseguire, se non non esiste esci*.

Quindi dobbiamo aggiungere qualcosa al codice per verificare se il file esiste, e lo possiamo fare in due modi, uno consiste nel creare una funzione che esegua il controllo e restituisca un valore Booleano True o False, oppure utilizzare un metodo già esistente nell'oggetto FileSystemObject che corrisponde al metodo *FileExists*. Utilizzando il metodo FileExists si può aggiungere un'istruzione *IF – Else* al codice attuale che prima di aprire il file Txt verifichi se il percorso è corretto. In pratica usando un'espressione del genere

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream
    If <pathisvalid> Then
   Set alex_1 = alex.OpenTextFile("C:\Test\prova.txt")
    alex_1.Close
    Else
    MsgBox "Il percorso del file non è valido.", vbCritical, vbNullString
    Exit Sub
    End If
End Sub
```

Dove *<pathisvalid>* è solo un segnaposto per l'espressione che deve controllare se il file esiste oppure no, in altre parole, se il percorso è valido, pertanto sostituiamo il segnaposto con il comando alex.FileExist ("C\Test\prova.txt") e modifichiamo il listato in questo modo

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream

    If alex.FileExists("C:\Test\prova.txt") Then
   Set alex_1 = alex.OpenTextFile("C:\Test\prova.txt")
    alex_1.Close
    Else
    MsgBox "Il percorso del file non è valido.", vbCritical, vbNullString
    Exit Sub
    End If
End Sub
```

Fig. 3

In alternativa, possiamo usare la funzione Dir per verificare se il file esiste in questo modo: *If Dir("C:\Test\prova.txt") <> vbNullString Then*. Se si vuole usare *Dir ()* vorrei suggerire di aggiungere una funzione di tipo booleano al codice, che restituisca un valore vero o falso se il percorso è corretta o meno. Converrete che ricevere un parametro di ritorno vero o falso sia quantomeno azzeccato all'operazione che si sta cercando di fare. Possiamo ora definire questa funzione

```
Function FileExist(filePath As String) As Boolean
    If Dir(filePath) <> vbNullString Then
        FileExist = True
    Else
        FileExist = False
    End If
End Function
```

Oppure con una sintassi diversa

```
Function fileExist(filePath As String) As Boolean
    fileExist = IIf(Dir(filePath) <> vbNullString, Vero, Falso)
End Function
```

Personalmente ritengo che non abbiamo assolutamente bisogno di quest'ultimo listato che utilizza il segnaposto (o placeholder) IIF, in quanto sappiamo che la funzione *FileExist* fa esattamente la stessa cosa, ma soprattutto vale la pena di sapere come gestire situazioni simili, creando le proprie funzioni e questo è un ottimo modo. E 'sempre una buona regola assegnare tutti i dati a variabili, se si memorizza il percorso in un'unica variabile nel codice, invece di ripeterlo in vari contesti, il codice diventerà più facile da gestire, in fase di test e di debug.

Cosa succede se il percorso o il nome del file cambia? Per non scorrere l'intero progetto alla ricerca di tutte le occorrenze di percorso sarebbe molto meglio e più facile da gestire se si usa una variabile che identifica il percorso del file e in caso di modifica si edita solo la variabile e il resto del codice rimane intatto.

Pertanto aggiungiamo una variabile per memorizzare il percorso, creando una variabile String denominata *filePath* a cui assegniamo il valore di "C:\Test\prova.txt"

```
Dim filePath As String
filePath = "C:\Test\prova.txt"
```

Da notare che la variabile filePath viene utilizzata nella Function *FileExist* e *OpenTextFile*, così ora il codice diventa:

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream
    Dim filePath As String
    filePath = "C:\Test\prova.txt"
    If alex.FileExists(filePath) Then
   Set alex_1 = alex.OpenTextFile(filePath)
    alex_1.Close
    Else
    MsgBox "Il percorso del file non è valido.", vbCritical, vbNullString
    Exit Sub
    End If
End Sub
```

Ora abbiamo costruito la procedura che apre il file se il percorso è corretto, oppure visualizza un messaggio di errore se il percorso del file non è corretto. Però questa procedura andrebbe bene se sul pc che viene utilizzato ci sia un solo utente che si logga all'apertura di Windows, oppure, se si sposta il percorso di ricerca sul desktop, servono altri parametri, oppure se ci fossero più utenti che si loggano sul pc. In questo caso verrebbe rimandato un errore di percorso non trovato a meno che non modifichiamo il listato usando una funzione che restituirebbe il nome attualmente loggato in Windows. Si tratta di usare la funzione [i[Environ ()[/i] che per essere usata si deve modificare il percorso del file in questo modo:

```
filePath = "C:\Users\" & Environ$("Username") & "\Test\prova.txt"
```

Se si esegue questo codice la macro apre e chiude con successo il file prova.txt, ma non possiamo avere nessuna certezza, pertanto possiamo aggiungere una dichiarazione temporanea al codice per verificare se il file è effettivamente aperto.

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream

    Dim filePath As String
    filePath = "C:\Users\" & Environ$("Username") & "\Desktop\prova.txt"

    If alex.FileExists(filePath) Then
   Set alex_1 = alex.OpenTextFile(filePath)
   Debug.Print IIf(alex_1 Is Nothing, "Il file è chiuso", "Il file è aperto")
    alex_1.Close
    Else
    MsgBox "Il percorso del file non è valido.", vbCritical, vbNullString
    Exit Sub
    End If
End SubFunction fileExist(filePath As String) As Boolean
    If Dir(filePath) <> vbNullString Then
       fileExist = True
    Else
       fileExist = False
    End If
End Function
```

Se il codice sopra esposto lo eseguiamo è possibile vedere se il file viene aperto oppure no, pertanto attiviamo la *Finestra Immediata* dal menu **Visualizza – Finestra immediata** oppure cliccando la combinazione di tasti **CTRL + G** e si dovrebbe vedere che il file viene aperto dopo aver eseguito la macro

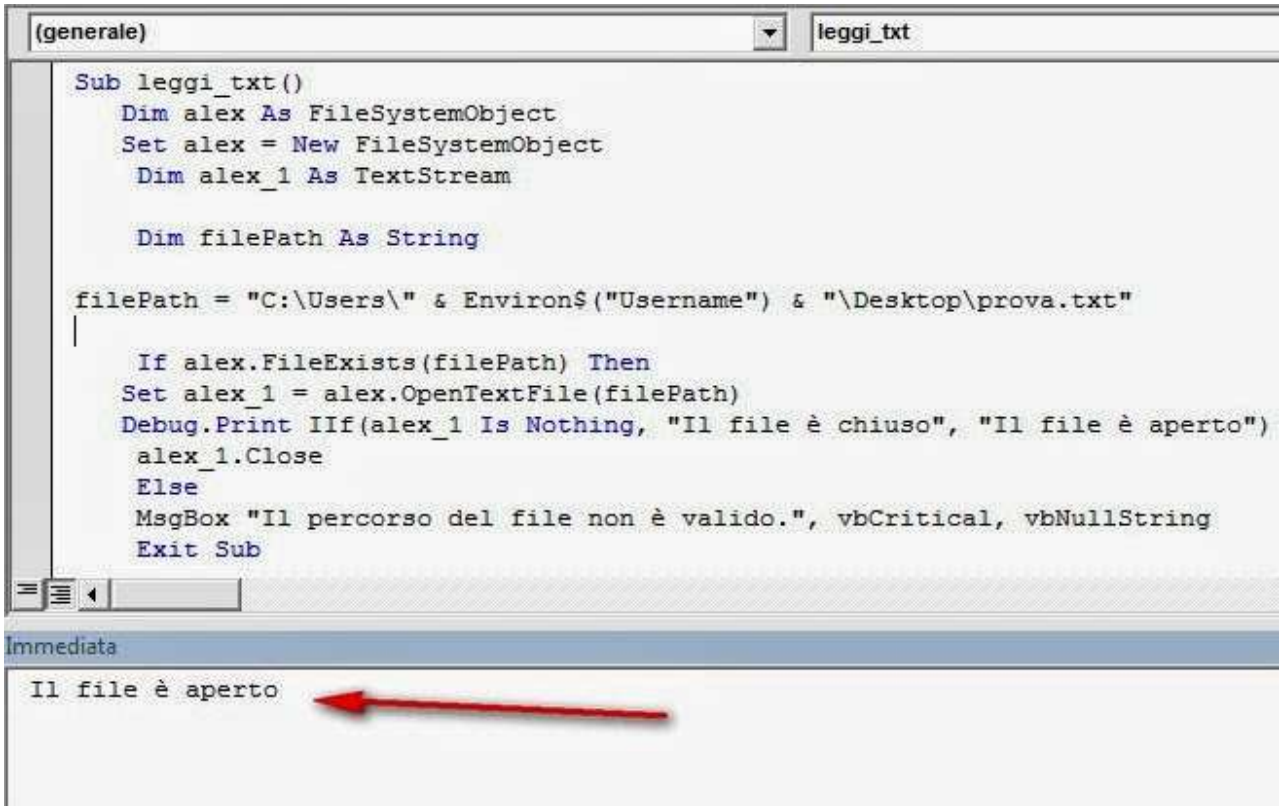


Fig. 4

Come si può vedere nella figura sopra riportata il codice sta funzionando benissimo, ma cosa accadrebbe se ci fosse un errore nel file prova.txt, oppure se fosse stato danneggiato, oppure se il metodo OpenTextFile non è riuscito a leggere il file. Certamente sarebbe apparso un messaggio minaccioso di errore di runtime.
Si dovrebbe sempre prendere in considerazione uno scenario simile se si lavora con i file, se qualcosa va storto nell'esecuzione del codice il programma si ferma immediatamente, così, tutti gli oggetti aperti, tutti i riferimenti sarebbe rimasti aperti e molto probabilmente non saremmo in grado di liberare la memoria a meno che il *garbage collection* (modalità automatica di gestione della memoria nei sistemi Windows) non ci fosse venuto in aiuto.

Questo è un caso in cui l'istruzione *On Error GoTo <label>* è molto utile, pertanto proviamo ad applicare il gestore degli errori al nostro esempio. Considerate le seguenti modifiche

```
Sub leggi_txt()
   Dim alex As FileSystemObject
   Set alex = New FileSystemObject
    Dim alex_1 As TextStream
    Dim filePath As String
    filePath = "C:\Users\" & Environ$("Username") & "\Desktop\prova.txt"

    If alex.FileExists(filePath) Then
    On Error GoTo Err
   Set alex_1 = alex.OpenTextFile(filePath)
    alex_1.Close
    Else
    MsgBox "Il percorso del file non è valido.", vbCritical, vbNullString
    Exit Sub
    End If

Err:
    MsgBox "Errore durante la lettura del file.", vbCritical, vbNullString
    alex_1.Close
    Exit Sub
End Sub
```

Se si verifica un errore durante la lettura del file l'istruzione *On goto errore Err* fermerà l'esecuzione del codice e salterà all'etichetta *Err* che mostrerà un messaggio di errore, per poi uscire dalla macro attualmente in esecuzione. Passiamo ora alla lettura del contenuto del file prova.txt. I tre approcci più comuni per leggere il contenuto di un file txt sono i metodi Read ReadAll e ReadLine applicabili all'oggetto TextStream.

Il metodo .ReadAll è molto facile da usare, se inseriamo una riga di codice dopo la chiamata OpenTextFile di questo tipo: *Range("A1") = alex_1.ReadAll*
Si recupera il contenuto del file prova.txt e lo si inserisce nella prima cella in alto a sinistra (A1) del foglio di calcolo attivo.

	A	B
	Questa è la riga uno questa invece è la seconda linea è questa è la linea n ° 3 ecco la quarta riga del file e finiamo con la Linea 5	
1		
2		

Fig. 5

Se si clicca sulla cella A1 e si osserva la barra della formula si sarà in grado di vedere meglio in quale formato avete ricevuto i vostri dati. Dovrebbe assomigliare a questa

A1

Questa è la riga uno
questa invece è la seconda linea
è questa è la linea n ° 3
ecco la quarta riga del file
e finiamo con la Linea 5

	A	B	C	D	E	F	G
	Questa è la riga uno questa invece è la seconda linea è questa è la linea n ° 3 ecco la quarta riga del file e finiamo con la Linea 5						
1							

Fig. 6

Come si può vedere il metodo *ReadAll* recupera tutti i dati da un file di testo in una volta nello stesso formato che sono disposti nel file di testo. E' ovvio che ricevere dei dati in questo formato non è un buon approccio e non consentono una manipolazione degli stessi, ma possiamo usare la funzione *Read*, la quale accetta un parametro che corrisponde al numero di caratteri che si desidera leggere. Se modifichiamo il codice, sostituendo *alex_1.ReadAll* con*alex_1.Read(5)*, verranno restituiti i primi 5 caratteri dal file di testo nella cella A1. Potete cambiare il valore 5 in qualsiasi altro numero e se si supera il numero di caratteri nel file di testo verrà recuperato l'intero file.

Anche questo metodo non presenta le caratteristiche desiderate per importare dei dati e manipolarli, non possiamo sapere quanti caratteri dobbiamo importare per avere dei dati di senso compiuto, se si tratta di dati in formato stringa oppure avere dei valori numerici che non vengano troncati. Esiste un approccio diverso per leggere il file di testo, usando l'istruzione *ReadLine* e come suggerisce il nome stesso permette di leggere il file riga per riga e possiamo con le adeguate istruzioni depositare i dati in celle separate. Se modifichiamo il codice in questo modo

```
Set alex_1 = alex.OpenTextFile(filePath)
    Do While Not alex_1.AtEndOfStream
    Range("A" & Range("A" & Rows.Count).End(xlUp).Row + 1) = alex_1.ReadLine
    Loop
    alex_1.Close
```

Il risultato sarà come il seguente

	A	B
1		
2	Questa è la riga uno	
3	questa invece è la	
4	è questa è la linea n ° 3	
5	ecco la quarta riga del	
6	e finiamo con la Linea 5	
7		

Fig. 7

Come si può notare la prima riga è stata lasciata vuota, che potrebbe ospitare le intestazioni dei campi, ma se si osserva l'istruzione *Range("A" & Range("A" & Rows.Count).End(xlUp).Row + 1)* , se si esegue il codice due volte, alla successiva esecuzione i dati verrebbero incollati sotto l'ultima riga utilizzata nella colonna A, in alternativa è possibile utilizzare una variabile per sovrascrivere i dati in questo modo:

```
Set alex_1 = alex.OpenTextFile(filePath)
   Dim i As Long
   i = 1
  Cells.ClearContents
    Do While Not alex_1.AtEndOfStream
   Range("A" & i) = alex_1.ReadLine
    i = i + 1
    Loop
    alex_1.Close
```

e si ottiene un risultato come il seguente:

	A	B
1	Questa è la riga uno	
2	questa invece è la	
3	è questa è la linea n ° 3	
4	ecco la quarta riga del	
5	e finiamo con la Linea 5	
6		

Fig. 8

Concludendo ormai si dovrebbe facilmente essere in grado di aprire qualsiasi file txt con codice VBA ed estrarre i dati da esso, vediamo il codice per intero nei due approcci presentati

```
Sub leggi_txt()
    Dim alex As FileSystemObject
    Set alex = New FileSystemObject
    Dim alex_1 As TextStream
    Dim filePath As String
    filePath = "C:\Users\" & Environ$("Username") & "\Desktop\prova.txt"
    If alex.FileExists(filePath) Then
        On Error GoTo Err
        Set alex_1 = alex.OpenTextFile(filePath)
   Dim i As Long
   i = 1
  Cells.ClearContents
    Do While Not alex_1.AtEndOfStream
   Range("A" & i) = alex_1.ReadLine
    i = i + 1
    Loop
    alex_1.Close
    Else
        MsgBox "Il percorso del file non è valido.", vbCritical, vbNullString
        Exit Sub
    End If
    Exit Sub
Err:
    MsgBox "Errore durante la lettura del file.", vbCritical, vbNullString
    alex_1.Close
    Exit SubEnd Sub

Function fileExist(filePath As String) As Boolean
    If Dir(filePath) <> vbNullString Then
        fileExist = True
    Else
        fileExist = False
    End If
End Function
```

Oppure

```
Sub leggi_txt()
    Dim alex As New FileSystemObject
    Dim alex_1 As TextStream
    Dim filePath As String
    filePath = "C:\Users\" & Environ$("Username") & "\Desktop\prova.txt"
    If Not fileExist(filePath) Then GoTo FileDoesntExist
    On Error GoTo Err
    Set alex_1 = alex.OpenTextFile(filePath)
    With Sheets(1).Range("A1")
        .ClearContents
        .NumberFormat = "@"
        .Value = alex_1.ReadAll
    End With
    alex_1.Close
    Exit Sub

FileDoesntExist:
    MsgBox "Il File non esiste", vbCritical, "File not found!"
    Exit Sub

Err:
    MsgBox "Errore durante la lettura del file.", vbCritical, vbNullString
    alex_1.Close
    Exit Sub
End Sub

Function fileExist(path As String) As Boolean
    fileExist = IIf(Dir(path) <> vbNullString, True, False)
End Function
```

Manipolare file e Cartelle in VBA

VBA fornisce alcuni metodi per lavorare con i file, ma usando le funzioni di base come Dir, Name etc. che presentano una stretta correlazione di comportamento con i comandi DOS, ma purtroppo riducono notevolmente il raggio d'azione, appesantendo notevolmente il listato del codice. Per estendere le possibilità per quanto riguarda la gestione di file e directory, Microsoft ha sviluppato una serie di oggetti raggruppati all'interno della libreria **Microsoft Scripting Runtime**, e questo insieme di classe gerarchica ha una sola radice: il **FileSystemObject**, più comunemente conosciuta come **FSO**.

Il modello File System Object (FSO) è uno strumento basato sugli oggetti per l'utilizzo di cartelle e file e consente di utilizzare la sintassi *oggetto.metodo* con un numeroso gruppo di proprietà, metodi ed eventi per l'elaborazione di cartelle e file e inoltre è possibile utilizzare le istruzioni e i comandi tradizionali di Visual Basic. Il FileSystemObject è una manna per tutti gli sviluppatori che utilizzano Visual Basic, in quanto semplifica il compito di trattare con qualsiasi tipo di input e output di file e per interagire con la struttura del file System stesso.

Piuttosto che ricorrere a complesse chiamate alla API Win32, questo oggetto consente alle applicazioni di creare, modificare, spostare ed eliminare cartelle, nonché di rilevare l'esistenza ed eventualmente la posizione di cartelle specifiche, inoltre è possibile ottenere informazioni sulle cartelle, quali il nome, la data di creazione e dell'ultima modifica e così via. Il modello FSO comprende i seguenti oggetti:

FileSystemObject	Consente di creare, eliminare e gestire unità, cartelle e file e di recuperare informazioni su tali elementi.
Drive	Consente di raccogliere informazioni relative a un'unità collegata al sistema, come la quantità di spazio disponibile e il nome di condivisione. Tenere presente che "drive" nel modello FSO non corrisponde solo al termine inglese per indicare il disco rigido: può trattarsi di un'unità CD-ROM, un disco RAM e così via.
Folder	Consente di creare, eliminare o spostare cartelle e di richiedere al sistema i loro nomi, i percorsi e altre informazioni.
File	Consente di creare, eliminare o spostare file e di richiedere al sistema i loro nomi, i percorsi e altre informazioni.
TextStream	Viene utilizzato per leggere e scrivere il contenuto di un file di testo

Fig. 1

Il modello FSO è contenuto nella libreria dei vari tipi di script, che si trova nel file Scrrun.dll e se non è già presente un riferimento alla libreria, è possibile crearne uno in questo modo: dal menu **Strumenti - Riferimenti**, e nella scheda che appare scorrere la lista e selezionare la voce **Microsoft Scripting Runtime** dall'elenco, quindi fare clic su **Ok**.

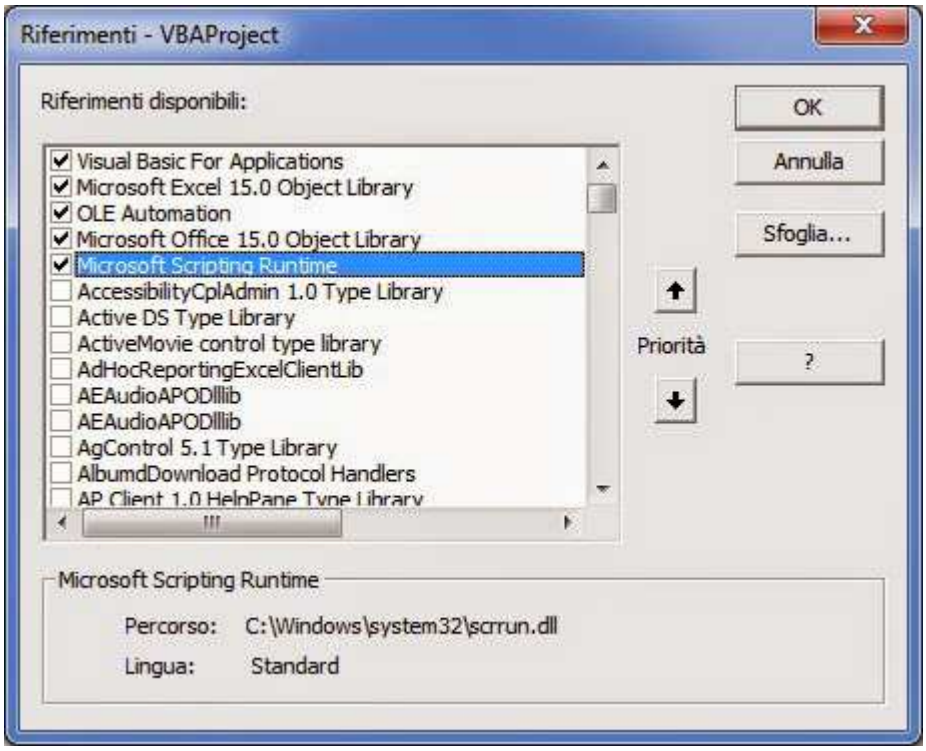


Fig. 2

E' possibile creare un oggetto FileSystemObject utilizzando il metodo CreateObject in questo modo:

```
oFSO = CreateObject("Scripting.FileSystemObject")
```

Oppure dimensionando una variabile come oggetto FileSystemObject, in questo modo:

```
Dim oFSO As New FileSystemObject
```

I metodi dell'oggetto FSO per le operazioni che può svolgere sono i seguenti:

Descrizione	Comando FSO
Creare un nuovo oggetto	**CreateFolder** o **CreateTextFile**
Eliminare un file o una cartella	**DeleteFile** o **File.Delete** oppure **DeleteFolder** o **Folder.Delete**
Copiare un oggetto	**CopyFile** o **File.Copy** oppure **CopyFolder** o **Folder.Copy**
Spostare un oggetto	**MoveFile** o **File.Move** oppure **MoveFolder** o **Folder.Move**
Accesso a un'unità, cartella o file esistente	**GetDrive**, **GetFolder** o **GetFile**

Fig. 3

Come si può notare, alcune funzioni del modello di oggetti FileSystemObject sono ridondanti, è possibile ad esempio copiare un file tramite il metodo *CopyFile* dell'oggetto FileSystemObject o tramite il metodo *Copy* dell'oggetto File. I metodi funzionano in modo identico, sono state esposte entrambe le versioni per offrire la massima flessibilità di programmazione. Non è necessario tuttavia utilizzare i metodi **Get** con i nuovi oggetti, in quanto se vengono usate le funzioni **Create** restituiscono già un puntamento a tali oggetti.

Se ad esempio viene creata una nuova cartella con il metodo CreateFolder, non è necessario utilizzare il metodo GetFolder per accedere alle sue proprietà, quali *Name*, *Path* o *Size*, è sufficiente impostare una variabile sulla funzione CreateFolder per ottenere un riferimento alla nuova cartella e quindi accedere alle sue proprietà, metodi ed eventi.

Esempio: Visualizzare le informazioni su un file utilizza alcune proprietà di FSO

```
Private Sub FileInfo(ByVal fileName As String)
    Dim fso As New FileSystemObject
    Dim fileSpec As File, mInfo As String
    Set fileSpec = fso.GetFile(fileName)
    mInfo = fileSpec.Name & vbCrLf
    mInfo = mInfo & "Creato il: "
    mInfo = mInfo & fileSpec.DateCreated & vbCrLf
    mInfo = mInfo & "Ultimo Accesso: "
    mInfo = mInfo & fileSpec.DateLastAccessed & vbCrLf
    mInfo = mInfo & "Ultima Modifica: "
    mInfo = mInfo & fileSpec.DateLastModified
    MsgBox mInfo, vbInformation, "Informazioni File"
    Set fileSpec = Nothing
End Sub
```

Che può essere richiamata con un codice come il seguente

```
Sub info1()
Dim infoF As String
infoF = "C:\Test\info1.txt"
FileInfo (infoF)
End Sub
```

Esempio: Visualizzare le informazioni su una cartella

```
Private Sub FolderInfo(ByVal folderName As String)
    Dim fso As New FileSystemObject
    Dim folderSpec As Folder, mInfo As String
    Set folderSpec = fso.GetFolder(folderName)
    mInfo = folderSpec.Name & vbCrLf
    mInfo = mInfo & "Creata il: "
    mInfo = mInfo & folderSpec.DateCreated & vbCrLf
    mInfo = mInfo & "Dimensione: "
    mInfo = mInfo & folderSpec.Size
    MsgBox mInfo, vbInformation, "Informazioni Cartella"
    Set folderSpec = Nothing
End Sub
```

Può essere richiamata in questo modo

```
Sub info2()
Dim infoC As String
infoC = "C:\Test"
FolderInfo (infoC)
End Sub
```

Esempio: Controllare se una cartella esiste

```
Function FolderE(DirName As String) As Boolean
On Error Resume Next
FolderE = GetAttr(DirName) And vbDirectory
End Function
```

La Function sopra riportata può essere testata usando un codice come il seguente

```
Sub Prova1()
Dim Percorso As String
Percorso = "C:\Test"
MsgBox FolderE(Percorso)
End Sub
```

Esempio: Controllare se un file esiste

```
Function FileE(FileName As String) As Boolean
On Error Resume Next
FileE = GetAttr(FileName) And vbArchive
End Function
```

Che può essere testata in questo modo

```
Sub Prova2()
Dim FileN As String
FileN = "C:\Test\info1.txt"
MsgBox FileE(FileN)
End Sub
```

Accedere a un disco

La collezione **Drives** dell'oggetto FileSystemObject fornisce l'accesso a tutti i record riconosciuti dal sistema operativo e ogni disco è quindi un oggetto ben distinto e il suo ID nella collezione è determinato dalla lettera di accesso (C, D, E, etc.), naturalmente, non si può scrivere, aggiungere o eliminare oggetti in questa collezione. Si deve tenere presente che usando Il metodo **DriveExists** della classe FileSystemObject è possibile determinare l'esistenza di un disco in base al suo nome.

```
Dim oFSO As Scripting.FileSystemObject
Dim oDrv As Scripting.Drive
Set oFSO = New Scripting.FileSystemObject
If oFSO.DriveExists("C") Then
   Set oDrv = oFSO.GetDrive("C")
Else
   MsgBox "Questo disco non esiste"
End If
```

Un'altra possibilità è quella di utilizzare il riferimento diretto alla raccolta **Drives**

```
Dim oFSO As Scripting.FileSystemObject
Dim oDrv As Scripting.Drive
Set oFSO = New Scripting.FileSystemObject
If oFSO.DriveExists("C") Then
   Set oDrv = oFSO.Drives("C")
Else
   MsgBox "Questo disco non esiste"
End If
```

Se il disco non esiste, i due metodi restituiranno lo stesso errore: *Errore # 5 chiamata di routine non valido*. Attenzione, che con questo metodo si accede ad un disco, o meglio in un disco, ma nel caso si tratti di un'unità CD-Rom e non fosse presente nessun CD non viene generato nessun errore, in sostanza non sappiamo se il dispositivo è accessibile. Esempio: Enumerare tutte le unità disco con un Loop For Each

```
Dim oFSO As Scripting.FileSystemObject
Dim oDrv As Scripting.Drive
Set oFSO = New Scripting.FileSystemObject
For Each oDrv In oFSO.Drives
    MsgBox oDrv.DriveLetter
Next oDrv
```

Le Proprietà del disco

- *DriveLetter:* E' la lettera utilizzato dal sistema operativo per accedere al disco.
- *DriveType:* Identifica il tipo di disco, CD-Rom, Disco Fisso, Ramdisk, etc.
- *FileSystem:* E' il tipo di file System presente nel disco (es.: NTFS)
- *AvailableSpace, FreeSpace:* Indica lo spazio disponibile e lo spazio libero in byte
- *IsReady:* E' rappresentato da un valore booleano che indica se l'unità è disponibile.
- *Path:* Indica il percorso del disco.
- *RootFolder:* Corrisponde alla cartella principale, e fornisce l'accesso a tutti i file presenti sul disco.
- *SerialNumber:* Indica il numero di serie del disco.
- *ShareName:* Restituisce una stringa corrispondente alla quota del disco. Questa stringa sarà nulla se il disco non è condiviso.
- *VolumeName:* Restituisce il nome del volume (non dell'unità) in una stringa. (es.: Dati)
- *TotalSize:* dimensioni del disco in byte

Alcuni esempi di funzioni per la gestione dei dischi, iniziando dal determinare il numero di CD-ROM (compresi quello virtuale) installato sul Pc:

```
Function leggiCD1() As Integer
Dim oFSO As Scripting.FileSystemObject
Dim oDrv As Scripting.Drive
Set oFSO = New Scripting.FileSystemObject
For Each oDrv In oFSO.Drives
    If oDrv.DriveType = CDRom Then leggiCD1 = leggiCD1 + 1
Next oDrv
End Function
```

Esempio: Restituire la lettera del disco rigido con più spazio disponibile

```
Function leggiD() As String
Dim oFSO As Scripting.FileSystemObject
Dim oDrv As Scripting.Drive
Dim intFree As Double
Set oFSO = New Scripting.FileSystemObject
'Percorso del disco
For Each oDrv In oFSO.Drives
    'Se si tratta di un disco rigido e se contiene un filesystem valido (formattato)
    If oDrv.DriveType = Fixed And oDrv.IsReady Then
        'Se lo spazio libero è superiore a intFree, allora sostituisci
       If oDrv.FreeSpace > intFree Then
            intFree = oDrv.FreeSpace
            leggiD = oDrv.DriveLetter
        End If
    End If
Next oDrv
End Function
```

Come si può vedere in questo esempio, la manipolazione di oggetti FSO fornisce un codice strutturato nello stesso modo come se si utilizza il metodo **DAO**(Database.OpenRecordset) in cui lo stesso oggetto viene richiamato più volte. Per questo motivo, è ampiamente raccomandato **il refactoring** dei blocchi di codice in questo modo.

```
Function leggiD1() As String
Dim oFSO As Scripting.FileSystemObject
Dim oDrv As Scripting.Drive
Dim intFree As Double
Set oFSO = New Scripting.FileSystemObject
'Percorso del disco
For Each oDrv In oFSO.Drives
    With oDrv
    'Se si tratta di un disco rigido e se contiene un filesystem valido (formattato)
        If .DriveType = Fixed And .IsReady Then
        'Se lo spazio libero è superiore a intFree, allora sostituisci
            If .FreeSpace > intFree Then
                intFree = .FreeSpace
                leggiD1 = .DriveLetter
            End If
        End If
End With
Next oDrv
End Function
```

Gestione Cartelle

Per accedere a una cartella si osserva lo stesso metodo usato per i dischi, in seguito vedremo i vari metodi e proprietà dell'oggetto cartella. L'accesso a un file istanziando un oggetto Folder, può essere fatto in due modi:

- Direttamente dall'oggetto FSO
- Dalla cartella principale

Prendiamo il caso di un accesso usando FSO e Il metodo **GetFolder (path)** che restituisce un oggetto *Folder*, corrispondente al percorso passato come parametro.

```
Dim oFSO As Scripting.FileSystemObject
Dim oFld As Folder
Set oFSO = New Scripting.FileSystemObject
Set oFld = oFSO.GetFolder("C:\Windows")
```

Se però la cartella non esiste, verrà restituito un *errore 76 (percorso non trovato)*, per cui è opportuno gestirlo in questo modo.

```
On Error GoTo err
Dim oFSO As Scripting.FileSystemObject
Dim oFld As Folder
Set oFSO = New Scripting.FileSystemObject
' simulazione errore
Set oFld = oFSO.GetFolder("C:\Windows0")
fine:
    Exit Function
err:
    If err.Number = 76 Then
        MsgBox "Questa cartella non esiste"
    Else
        MsgBox "Errore Sconosciuto"
    End If
    Resume fine
```

Si noti che l'errore 76 può anche essere evitato controllando l'esistenza del file dal FSO con il metodo **FolderExists** in questo modo

```
Set oFSO = New Scripting.FileSystemObject
If oFSO.FolderExists("C:\Windows0") Then
    Set oFld = oFSO.GetFolder("C:\Windows0")
Else
MsgBox "Questa cartella non esiste"
End If
```

Tuttavia, non vi è alcuna garanzia che il file non venga eliminato tra il test di verifica e il tentativo di accesso, per cui è fondamentale gestire l'errore 76 con l'argomento **On Error**, oppure usando un'altra tecnica che è quella di utilizzare la gerarchia delle cartelle nel file System, in cui ogni oggetto *Folder* dispone di una proprietà **SubFolders** per consolidare le sue sottocartelle. Nel caso di *C:\Windows*, dove Windows rappresenta un "figlio" della cartella C:\ (RootFolder)

```
Dim oFSO As Scripting.FileSystemObject
Dim oDrv As Drive, oFld As Folder
Set oFSO = New Scripting.FileSystemObject
Set oDrv = oFSO.GetDrive("C")
Set oFld = oDrv.RootFolder.SubFolders("Windows")
```

Anche se a prima vista il codice sembra più complesso, o più pesante, poiché l'accesso al disco è separato dal file, con questo metodo saremo in grado di conoscere il livello di errore in caso di mancata esecuzione, in altre parole, è l'unità C, che non è disponibile o inesistente su Windows? Possiamo gestirlo in questo modo:

```
On Error GoTo err
Dim oFSO As Scripting.FileSystemObject
Dim oDrv As Drive, oFld As Folder
Set oFSO = New Scripting.FileSystemObject
Set oDrv = oFSO.GetDrive("C")
Set oFld = oDrv.RootFolder.SubFolders("Windows")
fine:
   Exit Function
err:
      Select Case err.Number
         Case 5: MsgBox "Il disco non è disponibile"
         Case 76: MsgBox "Il record non esiste in questo disco"
         Case Else: MsgBox "Errore sconosciuto"
      End Select
   Resume fine
```

Come per l'accesso, è possibile utilizzare due diverse tecniche per creare una cartella:

- Utilizzando direttamente l'FSO
- Utilizzando un oggetto Folder tramite la sua collezione SubFolders

Usando FSO si può fare con questo codice:

```
On Error GoTo err
Dim oFSO As Scripting.FileSystemObject
Dim oDrv As Drive, oFld As Folder
Set oFSO = New Scripting.FileSystemObject
Set oFld=oFSO.CreateFolder ("C:\Test")
fine:
   Exit Function
err:
      Select Case err.Number
         Case 58: MsgBox "Il file esiste già"
         Case 76: MsgBox "Percorso non corretto"
         Case Else: MsgBox "Errore sconosciuto"
      End Select
   Resume fine
```

Se il percorso non è valido, o la directory principale del disco è inesistente, viene generato un errore 76 (percorso non trovato), mentre se il file esiste già, l'operazione non riesce e rimanda un errore 58 (File già esistente). L'oggetto *Ofld* restituito dal metodo *CreateFolder* è riutilizzabile immediatamente dopo il codice, dal momento che la collezione *SubFolders* viene usata in questo modo:

```
On Error GoTo err
Dim oFSO As Scripting.FileSystemObject
Dim oDrv As Drive
Set oFSO = New Scripting.FileSystemObject
Set oDrv = oFSO.GetDrive("C")
oDrv.RootFolder.SubFolders.Add ("Test")
fine:
   Exit Function
err:
      Select Case err.Number
         Case 5: MsgBox "Il disco non è disponibile"
         Case 58: MsgBox "Il file esiste già"
         Case 76: MsgBox "Percorso non corretto"
         Case Else: MsgBox "Errore sconosciuto"
      End Select
   Resume fine
```

Come si può vedere dal codice, la differenza sta solo nella gestione degli errori. Per chi ha familiarità con DAO, probabilmente si ricorderà la proprietà Attributes per oggetti diversi che rappresenta l'aggiunta logica dei diversi valori degli elementi di qualificazione. Ad esempio, una cartella potrebbe essere nascosta, oppure nascosta e archiviata, etc. I valori possibili sono:

- *Normal*
- *ReadOnly*: Sola lettura
- *Hidden*: Cartella nascosta
- *System*: Cartella Sistema
- *Archive*: archivio di file
- *Compressed*: Cartella compressa

Di seguito sono riportati alcuni possibili listati di prova:

```
If oFld.Attributes And Directory Then MsgBox "Nascosto"
If oFld.Attributes And (Hidden + ReadOnly) Then MsgBox "Nascosta e di sola lettura"
```

La proprietà Attributes può essere applicata in lettura o in scrittura, il che significa che è possibile modificare gli attributi di cartella. Esempio per rimuovere una modalità cartella nascosta:

```
If oFld.Attributes And Hidden Then
 oFld.Attributes = oFld.Attributes - Hidden
```

Le proprietà dell'oggetto Folder

- *Attributes*: Come visto sopra, attribuisce la cartella.
- *DateCreated*: Data di creazione della cartella
- *DateLastAccessed*: Data dell'ultimo accesso alla cartella
- *DateLastModified*: Data ultima modifica
- *Drive*: Corrispondente al disco in cui si trova la cartella
- *Files*: E' una raccolta di file nella cartella
- *IsRootFolder*: E' un valore booleano che determina se la cartella è la radice del disco
- *Name*: Indica il nome della cartella
- *ParentFolder*: Folder corrisponde alla cartella principale, se la cartella è una cartella RootFolder questa proprietà restituisce Nothing.
- *Path*: Indica il percorso completo della cartella
- *ShortName*: E' il nome "breve" di una cartella con un massimo di 8 caratteri
- *ShortPath*: percorso completo della cartella in cui ogni componente è conforme alla norma ShortName
- *Size*: Dimensione totale del file in byte. Questa è la somma delle dimensioni di tutti i file nella cartella e relative sottocartelle.
- *SubFolders*: Indica un raggruppamento di sottocartelle
- *Type*: Tipo di file. In tutti i casi esaminati, è FileFolder

Il metodo Copy dell'oggetto Folder

Il metodo Copy consente di copiare la cartella e il suo contenuto in un altro percorso (esistenti o meno). Sintassi: *Copy(Destination As String, [OverWriteFiles As Boolean = True])*. Dove *Destination* rappresenta un percorso valido di destinazione della copia e se *OverWriteFiles* è vero, i file presenti nella directory di destinazione vengono sovrascritti se hanno lo stesso nome. Esempio: *oFld.Copy "C:\Test", True*

Se il percorso di destinazione non è corretto, viene generato un errore 76 (percorso non trovato) e se OverWriteFiles è False e la destinazione contiene già dei file con lo stesso nome, viene generato un errore 58 (File già esistente). Il metodo CopyFolder dell'oggetto FSO riproduce lo stesso comportamento. Esempio: *oFSO.CopyFolder("C:\Test","C:\Test2",True)*

Il metodo Delete dell'oggetto Folder

Il metodo Delete rimuove la cartella specificata. Sintassi: *Delete([Force As Boolean = False])*. Il parametro Force se viene collocato a True esegue una forzatura per eliminare file di sola lettura nella cartella specificata e relative sottocartelle. Se invece Force è False, viene generato un errore 70 (Permesso negato) e il file diventa di sola lettura, e vale anche se il file è aperto. Esempio: *oFld.Delete False*

E 'possibile utilizzare un altro metodo per eliminare una cartella usando il metodo DeleteFolder dell'oggetto FileSystemObject. Esempio: *oFSO.DeleteFolder "C:\Test", False*

Il metodo Move dell'oggetto Folder

Il metodo Move sposta la cartella di destinazione specificata in un altro percorso.
Sintassi: *Move(Destination As String)*. Esempio: *oFld.Move "C:\Test2"*

E 'possibile spostare una cartella in un'altra, tuttavia, inoltre se il contenuto esiste già nella destinazione verrà generato un errore 58. Anche in questo caso, si può utilizzare l'equivalente FSO con il metodo MoveFolder. Esempio: *oFSO.MoveFolder "C:\Test", "C:\Testi2"*

Cartelle speciali

Il metodo *GetSpecialFolder* consente l'accesso a directory specifiche.
Sintassi: *GetSpecialFolder(SpecialFolder As SpecialFolderConst) As Folder* e i valori possibili per SpecialFolder sono:

- *WindowsFolder*: Cartella in cui è installato Windows
- *SystemFolder*: Cartella di Sistema (Windows)
- *TemporaryFolder*: Cartella per memorizzare i file temporanei

Gestione dei file

Questa è la gerarchia più bassa dell'elemento e ogni file è rappresentato da un oggetto file nell'insieme Files in un oggetto Folder. Per accedere a un file si possono essere utilizzati due metodi per restituire un oggetto File.

- Utilizzando il metodo GetFile del FileSystemObject
- Utilizzando la collezione File di un oggetto Folder

Usando FSO può essere espresso in questo modo

```
On Error GoTo err
Dim oFSO As Scripting.FileSystemObject
Dim oFl As Scripting.File
Set oFSO = New Scripting.FileSystemObject
Set oFl = oFSO.GetFile("C:\Test\info1.txt")
fine:
   Exit Function
err:
      Select Case err.Number
         Case 53: MsgBox "Il file non è stato trovato"
         Case Else: MsgBox "Errore sconosciuto"
      End Select
   Resume fine
```

Si deve prestare attenzione che se il percorso del file non è corretto, viene generato l'errore 53 (File non trovato). In questi casi viene usato il *metodoFileExists* per verificare l'esistenza del file.

```
On Error GoTo err
Dim oFSO As Scripting.FileSystemObject
Dim oFl As Scripting.File
Set oFSO = New Scripting.FileSystemObject
If oFSO.FileExists("C:\Test\info1.txt") Then
   Set oFl = oFSO.GetFile("C:\Test\info1.txt")
End If
fine:
   Exit Function
err:
      Select Case err.Number
         Case 53: MsgBox "Il file non è stato trovato"
         Case Else: MsgBox "Errore Sconosciuto"
      End Select
   Resume fine
```

Mentre usando l'oggetto Folder il codice è il seguente:

```
On Error GoTo err
Dim oFSO As Scripting.FileSystemObject
Dim oFld As Scripting.Folder
Dim oFl As Scripting.File
Set oFSO = New Scripting.FileSystemObject
Set oFld = oFSO.GetFolder("C:\Test")
Set oFl = oFld.Files("info1.txt")
fine:
   Exit Function
err:
      Select Case err.Number
         Case 76: MsgBox "La cartella non esiste"
         Case 53: MsgBox "Il file non si trova in questa cartella"
         Case Else: MsgBox "Errore Sconosciuto"
      End Select
   Resume fine
```

Le proprietà dell'oggetto File

- *Attributes*: Gli attributi dei file, stessa proprietà per le cartelle.
- *DateCreated*: Data di creazione del file
- *DateLastAccessed*: Data dell'ultimo accesso
- *DateLastModified*: Data ultima modifica
- *Drive*: Indica l'unità corrispondente al disco in cui risiede il file
- *Name*: Nome del file.
- *ParentFolder*: E' la Cartella che contiene il file.
- *Path*: Indica il percorso completo del file.
- *ShortName*: Denominazione rispettando lo standard 8.3 (8 caratteri per il nome e 3 per l'estensione).
- *ShortPath*: Percorso completo della cartella in cui ogni componente è conforme alla norma ShortName
- *Size*: Dimensione in byte del file
- *Type*: Tipo di file

Il metodo Copy dell'oggetto File

Il metodo copy permette di copiare il file in un'altra destinazione. Sintassi: *Copy(Destination As String, [OverWriteFiles As Boolean = True])*

Dove *Destination* è un percorso valido del file copiato, e se esiste già un file con lo stesso nome, verrà sovrascritto con l'unica condizione che OverWriteFiles sia uguale a true. Esempio: *oFl.Copy "C:\Test2\info1.txt", True*

Se il percorso di destinazione non è corretto, viene generato un *errore 76* (percorso non trovato). Se invece *OverWriteFiles* è uguale a False e il file esiste già, viene generato un *errore 58* (File già esistente). Il metodo CopyFile dell'oggetto FSO riproduce lo stesso comportamento. Esempio: *oFSO.CopyFile("C:\Test\info1.txt","C:\Test2\info1. txt ",True)*

Il metodo Delete dell'oggetto File

Il metodo Delete elimina il file specificato. Sintassi: *Delete([Force As Boolean = False])*, dove il parametro *Force* se posto uguale a True, forza l'eliminazione di un file anche se è di sola lettura, se invece è uguale a False e il file è di sola lettura, viene generato un *errore 70* (Autorizzazione negata). Questo errore viene generato se il file è aperto. Esempio: *oFl.Delete False*. È anche possibile utilizzare il metodo DeleteFile per l'oggetto FSO. Esempio: *oFSO.DeleteFile "C:\Test\info1.txt", False*

Il metodo Move dell'oggetto File

Il metodo Move permette di spostare il file in un'altra destinazione. Sintassi: *Move(Destination As String)*. Esempio: *oFl.Move "C:\Test2\info1.txt"*. In alternativa, si può usare il metodo MoveFile dell'oggetto FileSystemObject. Esempio: *oFSO.MoveFile "C:\Test\info1.txt ", "C:\Test2\info1.txt "*

Questi metodi possono essere usati anche per rinominare un file.
Esempio: *oFSO.MoveFile "C:\Test\info1.txt ", "C:\Test\info1_old.txt "*

Verificare l'esistenza di un percorso

Sebbene il FileSystemObject fornisce diversi metodi per verificare l'esistenza di un file, è difficile in caso di errore sapere quale parte del percorso ha causato il problema in caso di errore. Immaginate questo percorso: *C:\Test\Test2\Prove3* dove Prove3 non esiste. Usando il metodo FolderExists viene restituito il valore di False, ma l'utente ignorerà se è stato causato da Test, Test2 o Prove3. In questo caso si può usare una funzione più completa per identificare il file alla radice dell'errore.

```
Function TestC(strC As String) As Boolean
On Error GoTo err
Dim oFSO As Scripting.FileSystemObject, oFld As Scripting.Folder
Dim oDrv As Scripting.Drive, i As Integer
Dim strD() As String
'istanziare FSO
Set oFSO = New Scripting.FileSystemObject
'Accedere al disco
Set oDrv = oFSO.Drives(oFSO.GetDriveName(strC))
'Instanziare la cartella principale
Set oFld = oDrv.RootFolder
'tagliare il percorso della cartella
strD = Split(strC, "\")
'tentativi di accedere alle sotto cartelle
For i = 1 To UBound(strD) - 1
   Set oFld = oFld.SubFolders(strD(i))
Next i
TestC = True
fine:
   Exit Function
err:
   Select Case err.Number
      Case 5: MsgBox "Il disco non esiste"
      Case 76: MsgBox "Impossibile trovare il file: " & strD(i)
      Case Else: MsgBox "Errore Sconosciuto"
   End Select
   Resume fine
End Function
```

Metodi di elaborazione dei file con VBA

L'elaborazione dei file è la capacità di memorizzare i valori di un documento nel computer in modo da poter recuperare tali valori successivamente, oppure è la possibilità di salvare i valori da un'applicazione e recuperare tali valori quando è necessario. Prima di eseguire l'elaborazione dei file, la prima azione che è necessario eseguire consiste nel creare un file e per sostenere tale processo, VBA fornisce una procedura denominata Open che presenta la seguente sintassi

Open pathname For Output [Access access] [lock] As [#]filenumber [Len=reclength]

La dichiarazione **Open** mostra molti argomenti, alcuni sono necessari e altri no, le espressioni obbligatorie sono: La dichiarazione Open, l'espressione For Output e la dichiarazione As #, oltre all'argomento, pathname (percorso) che è rappresentato da una stringa che può essere il nome del file, inoltre Il file può avere un'estensione o meno. Ecco un esempio:

Open "Prova.txt"

Se si specifica solo il nome del file, sarebbe considerato nella stessa cartella in cui è la cartella di lavoro corrente (la cartella di lavoro che è stata aperta quando è stata eseguita questa istruzione). Se si desidera, è possibile fornire un percorso completo per il file, che include l'unità, il nome della cartella, fino al nome del file, con o senza estensione. Oltre al nome del file o il suo percorso, la modalità è un fattore necessario in quanto indica l'azione reale che si desidera eseguire, come la creazione di un nuovo file o solo l'apertura di uno esistente. Questo fattore può essere rappresentato da una delle seguenti parole chiave

- Output: verrà creato il file e pronto a ricevere i valori (normali)
- Binary: verrà creato il file e pronto a ricevere i valori in formato binario (come combinazioni di 1 e 0)
- Append: Se il file esiste già, verrà aperto e nuovi valori possono essere aggiunti alla fine

Ecco un esempio di creazione di un file:

```
Sub apri ()
   Open "Prova.txt" For Output As #1
End Sub
```

Il tipo di accesso al file è facoltativo, questo argomento specifica quali tipi di azioni saranno eseguite sul file, come ad esempio la scrittura dei valori o solo la lettura dei valori esistenti. Questo fattore può avere uno dei seguenti valori:

- Scrittura: Dopo che è stato creato un nuovo file, verranno scritti i nuovi valori
- Lettura e Scrittura: Quando un nuovo file è stato creato o un file esistente è stato aperto, i valori possono essere letti o scritti

Se si decide di specificare il tipo di accesso, si deve precedere il suo valore con la parola chiave **Access** mentre il fattore di **Lock** (Blocco) è facoltativo, e indica come il processore dovrebbe comportarsi mentre si utilizza il file. I suoi possibili valori sono:

- Shared (In comune): Altre applicazioni (effettivamente richiamati dal processo) possono accedere al file mentre l'applicazione corrente lo sta utilizzando
- Blocca Scrittura: Non lasciare che altre applicazioni (processi) possano accedere a questo file, mentre l'applicazione corrente (processo) sta scrivendo su di esso
- Blocca Lettura e Scrittura: Non permettere ad altre applicazioni (processi) di accedere a questo file, mentre l'applicazione corrente (processo) lo sta utilizzando

Sul lato destro dell'espressione **#**, si deve digitare un numero, per l'argomento *filenumber*, compreso tra 1 e 511 e se si lavora su un file solo, si consiglia di utilizzare il numero 1, mentre se si sta lavorando su più file, è necessario utilizzare un numero incrementale. Se non si è riuscito a tenere traccia del numero o esiste la possibilità di confondersi, per conoscere il numero successivo è possibile utilizzare, la funzione FreeFile (), che restituisce il numero successivo disponibile nella sequenza. L'argomento reclength è facoltativo, se il file è stato aperto, questo fattore specifica la lunghezza del record che è stato letto.

Chiusura di un file

Quando si crea e si inizia ad usare un file, o dopo aver aperto un file e mentre lo si utilizza, si impegnano risorse di sistema che potrebbero essere significative, ed è necessario quando si è terminato di utilizzare il file, liberare la memoria che si stava usando e rilasciare le risorse impegnate. Per eseguire questa operazione VBA fornisce una procedura denominata **Close**, la cui sintassi è:

Close [filenumberlist]

Dove l'argomento *filenumberlist* è lo stesso argomento *filenumber* che è stato precedentemente utilizzato per creare o aprire il file. Ecco un esempio di chiusura di un file:

```
Sub Prova ()
   Open "Prova.txt" For Output As #1
   Close #1
End Sub
```

Scrittura in un file

Dopo aver creato un file, si deve scrivere dei valori al suo interno, a sostegno di questa azione, VBA fornisce due procedure, una si chiama **Print** e la sua sintassi è:

Print #filenumber, [outputlist]

La sintassi della dichiarazione Print richiede due argomenti, ma è necessario solo *filenumber* che corrisponde all'argomento *filenumber* che si è usato per creare il file. Filenumber è seguito da una virgola e poi dall'argomento outputlist che può essere costituito da 0 a 1 o più valori. Questo argomento è facoltativo, se non si vuole assegnare un valore al file, si può lasciare vuoto. Se invece volete assegnare un valore, si deve digitare una virgola dopo filenumber e seguire queste regole:

- Se si desidera assegnare un valore con spazi vuoti, utilizzare la funzione **Spc ()** e passare un numero intero, tra parentesi che rappresenta il numero di spazi vuoti. Per esempio **Spc (4)** includerebbe 4 spazi vuoti. Questo argomento è facoltativo, il che significa che è possibile ometterlo
- Invece di un determinato numero di spazi vuoti, è possibile lasciare che il sistema operativo si occupi di questa operazione e per fare questo, si deve richiamare la funzione **Tab ()** come parte del fattore di outputlist. La funzione Tab () specifica il numero di colonne da includere prima del valore e può essere utile se si desidera l'allineamento dei valori che si scriveranno nel file. Questo argomento è facoltativo, il che significa che è possibile ometterlo
- Per scrivere una stringa deve essere racchiusa tra virgolette
- Per scrivere un numero, sia un numero intero, o decimale, è sufficiente inserire il numero normalmente
- Per scrivere un valore booleano, si deve inserire come **True** o **False**
- Per scrivere un valore di data o ora, si deve digitarlo tra **#** e **#** e seguire le regole di data o ore della lingua del sistema operativo installato
- Per scrivere un valore nullo, digitare **Null**

Ecco un esempio di scrittura di alcuni valori:

```
Sub Prova ()
   Open "Prova.txt" For Output As #1
   Print #1, "Gino"
   Print #1, "Gigetto"
   Print #1, True
   Print #1, #17/08/2014#
   Close #1
End Sub
```

Invece di scrivere un valore per riga, è possibile scrivere più di un valore con una sola dichiarazione. Per fare questo, si deve separarli con un punto e virgola o uno spazio vuoto. Ecco un esempio.

```
Sub Prova ()
   Open "Prova.txt" For Output As #1
      'I valori sono separate dal punto e virgola
   Print #1, "Gino"; "Gigetto"
     ' I valori sono separate da uno spazio vuoto
   Print #1, True #17/08/2014#
   Close #1
End Sub
```

La scrittura di un file

Oltre alla procedura Print, VBA fornisce anche una procedura denominata **Write** che può essere utilizzata per scrivere uno o più valori in un file. La sintassi è la stessa di quella dell'argomento Print:

Write #filenumber, [outputlist]

Dove l'argomento *filenumber* è necessario e deve essere specificato al momento della creazione del file, mentre l'argomento outputlist è facoltativo e se si vuole omettere, si deve digitare una virgola dopo la dichiarazione filenumber e terminare la dichiarazione di scrittura.

In questo caso, una riga vuota verrà scritta nel file. Per scrivere i valori nel file, si devono seguire queste regole.

- Per assegnare un valore con spazi vuoti, si deve richiamare la funzione **Spc ()** e passare un numero che rappresenta il numero di spazi vuoti. Questo fattore è facoltativo, il che significa che è possibile ometterlo
- Per assegnare il valore di un determinato numero di colonne, si deve richiamare la funzione **Tab ()** e passare il numero di colonne come argomento. Questo fattore è facoltativo, il che significa che è possibile ometterlo
- Per scrivere una stringa, si deve includerla tra virgolette
- Per scrivere un numero, si può inserirlo normalmente
- Per scrivere un valore booleano, si deve inserirlo TRUE o FALSE
- Per scrivere un valore nullo, inserire NULL
- Per scrivere un valore di data o ora, si deve inserirlo tra # e #

Ecco un esempio di scrittura di alcuni valori:

```
Sub Prova ()
   Open "Prova.txt" For Output As #1
   Write #1, "Gino"
   Write #1, "Dott."
   Write #1, "Ginetto"
   Write #1, #17/08/2014#
   Write #1, 12.30
   Write #1, True
   Close #1
End Sub
```

È anche possibile scrivere i valori sulla stessa linea, basta separarli con uno spazio vuoto, una virgola o un punto e virgola. Ecco un esempio:

```
Sub Prova ()
   Open "Prova.txt" For Output As #1
    ` I valori sono separate da un punto e virgola
   Write #1, "Gino"; "Dott."; "Ginetto"
   ` I valori sono separate da uno spazio vuoto
   Write #1, #17/08/2014#, 12.30
   Write #1, True
   Close #1
End Sub
```

Apertura di un file

A volte è necessario aprire un file esistente per poter leggere o scrivere altri dati, in questo caso VBA fornisce una procedura denominata **Open**, la cui sintassi è.

Open pathname For Input [Access access] [lock] As [#]filenumber [Len=reclength]

La sintassi della procedura Open mostra molti argomenti, alcuni sono necessari e altri no. Sono da ritenersi obbligatori i seguenti argomenti: La dichiarazione Open, l'espressione As # e il percorso che può essere il nome del file, inoltre il file può avere un'estensione o meno. Ecco un esempio:

Open "Prova.txt"

Se si specifica solo il nome del file, si cerca il file nella stessa cartella in cui è alloggiata la cartella di lavoro corrente. Se si desidera, è possibile fornire un percorso completo, ciò include l'unità, la cartella (opzionale), fino al nome del file, con o senza estensione. Oltre al nome del file o il suo percorso, è necessario fornire il tipo di accesso che si desidera fare. Per aprire un file, le modalità possono essere:

- **Binary**: Il file verrà aperto e il suo valore viene letto come valore binario
- **Append**: Il file verrà aperto e i nuovi dati verranno aggiunti alla fine dei dati già presenti
- **Input**: Il file verrà aperto in sola lettura
- **Random**: Il file sarà aperto per l'accesso casuale

Ecco un esempio di apertura di un file:

```
Sub Prova ()
   Open "Prova.txt" For Input As #1
End Sub
```

La modalità di accesso è facoltativa e può avere uno dei seguenti valori:

- Lettura: Dopo che il file è stato aperto, i valori presenti al suo interno verranno letti
- Lettura e Scrittura: Se il file è stato creato o aperto, i valori possono essere letti e/o scritti

Se si decide di specificare la modalità di accesso si deve precedere il suo valore con la parola chiave di accesso

La lettura da un file

Dopo l'apertura di un file, è possibile leggere i valori al suo interno, e prima di leggere il valore, si dovrebbe dichiarare una o più variabili che riceverebbero i valori da leggere. Si deve ricordare che i benefici utilizzando una variabile sono di riservare uno spazio di memoria in cui è possibile memorizzare un valore. Allo stesso modo, quando si legge un valore da un file, si otterrebbe il valore dal file e quindi memorizzare tale valore nella memoria del computer. Una variabile renderebbe più facile per fare riferimento a tale valore in caso di necessità.

Per sostenere la possibilità di aprire un file, il VBA fornisce due procedure. Se i valori sono stati memorizzati usando la dichiarazione Print, si possono leggere i valori, utilizzando la dichiarazione **Input** o **Line Input**. La sintassi della procedura è

Input #filenumber, varlist

La dichiarazione richiede due argomenti, ma il secondo può essere realizzato in varie parti. L'argomento filenumber è lo stesso che si è usato per aprire il file ed è seguito da una virgola. L'argomento *varlist* può essere di 1 o più parti. Per leggere un solo valore, dopo la virgola di filenumber, si deve digitare il nome della variabile che riceverà il valore. Ecco un esempio:

```
Sub Prova ()
   Dim primo As String
   Open "Prova.txt" For Input As #1
   Input #1, primo
   Close #1
End Sub
```

Allo stesso modo, è possibile leggere ogni valore su una riga separata. Uno dei migliori usi di usare la dichiarazione Input è la capacità di leggere molti valori utilizzando una singola istruzione. Per effettuare questa operazione, si devono digitare le variabili sulla stessa, separandole con virgole. Ecco un esempio:

```
Sub Prova ()
   Dim primo As String
   Dim secondo As String
   Dim tempo As Boolean
   Open "Prova.txt" For Input As #1
   Input #1, primo, secondo, tempo
   Close #1
End Sub
```

Se si dispone di un file che contiene molte linee, per leggere una riga alla volta, è possibile utilizzare la parola chiave Line Input. La sua sintassi è:

Line Input #filenumber, varname

Questa affermazione necessita di due argomenti, ed entrambi sono necessari. L'argomento filenumber è il numero che si sarebbe usato per aprire il file. Quando l'istruzione Line Input viene richiamata chiamato, legge una riga di testo fino a raggiungere la fine del file. Uno dei limiti di Line Input è la difficoltà a leggere qualcosa di diverso testo perché potrebbe non essere in grado di determinare dove termina la linea.

Nel riesaminare la possibilità di scrivere i valori in un file, abbiamo visto che l'istruzione Print scrive un valore booleano come Vero o Falso, se si utilizza l'istruzione Input per leggere tale valore, l'interprete potrebbe non essere in grado di leggere il valore. Abbiamo visto che in alternativa alla dichiarazione Print si può usare la dichiarazione Write, inoltre abbiamo visto che, tra le differenze tra Input e Write, quest'ultimo scrive i valori booleani con il simbolo #, questo rende possibile per l'interprete di leggere facilmente tale valore. Per queste ragioni, nella maggior parte dei casi, può essere più utile usare la scrittura quando si creano valori stringa diversi in un file.

Oggetti e Collezioni nei moduli di classe

Una raccolta è una serie di elementi in cui ogni elemento ha le stesse caratteristiche, in altre parole, tutti gli elementi possono essere descritte nello stesso modo. In programmazione, una collezione è una serie di elementi in cui tutti gli elementi condividono le stesse **proprietà** e **metodi**, se presenti. Ad esempio, una raccolta può essere fatta dai dipendenti di una società in cui ogni dipendente può essere descritto con le stesse caratteristiche ad esempio il nome.

Creazione di una collezione

Per supportare le collezioni, il linguaggio Visual Basic è dotato di una classe denominata **Collection**, in realtà, la Collezione di classe che ci accingiamo a studiare è quella definita in VBA e può essere usata per creare una collezione. Per fare questo, si deve dichiarare una variabile di tipo Collection. Ecco un esempio:

```
Sub Test()
   Dim dipendenti As Collection
End Sub
```

Dopo aver dichiarato la variabile, si deve allocare lo spazio in memoria per la stessa, utilizzando la parola chiave **Set** per assegnare una nuova istanza alla variabile. Ecco un esempio:

```
Sub Test()
   Dim dipendenti As Collection
   Set dipendenti = New Collection
End Sub
```

Invece di creare sempre una nuova collezione, a meno che non si debba, VBA è dotato di molte collezioni standard integrate, in modo da evitare di creare sempre la propria collezione e sono indicate come le collezioni incorporate.
Le classi Collection incorporate sono derivati dalla classe Collection di Visual Basic e hanno tutte le loro funzionalità primarie ereditate dalla collezione di classe. Questo significa anche che tutto ciò che menzioneremo per la raccolta classe si applica a qualsiasi raccolta incorporata. Per utilizzare una collezione precostituita, è possibile dichiarare una variabile. Ecco un esempio:

```
Sub Test()
   Dim CurrentSheets As Worksheets
End Sub
```

In realtà, quando Microsoft Excel viene avviato, la maggior parte (se non tutte) delle classi sono già disponibili in modo che non c'è bisogno di dichiarare la loro variabile prima di utilizzarle.

Caratteristiche, e operazioni su, una raccolta

L'operazione primaria da effettuare su una collezione consiste nell'aggiungere elementi, a sostegno di questa, la Collezione classe è dotata del metodo **Add**. La sua sintassi è:

```
Public Sub Add( _
  ByVal Item As Object, _
  Optional ByVal Key As String, _
  Optional ByVal { Before | After } As Object = Nothing _
)
```

Questo metodo richiede tre argomenti, ma solo il primo è necessario e la voce "argomento" specifica l'oggetto da aggiungere alla collezione. Ecco un esempio:

```
Sub Test()
    Dim dipendenti As Collection
    Set dipendenti = New Collection
    dipendenti.Add "Patrizia"
End Sub
```

Allo stesso modo, è possibile aggiungere il numero di elementi che si desidera:

```
Sub Test()
    Dim dipendenti As Collection
    Set dipendenti = New Collection
dipendenti.Add ”Patrizia”
dipendenti.Add    Giacomo”
dipendenti.Add ”Alice”
dipendenti.Add ”Franco”
End Sub
```

Ricordate che se si sta utilizzando una delle classi standard precostituite, non è necessario dichiarare una variabile, si può solo richiamare il metodo **Add** per aggiungere un elemento. Ecco un esempio:

```
Sub Test()
    Worksheets.Add
End Sub
```

Accesso a un elemento in una raccolta

Le voci di una raccolta sono organizzate in una sequenza in cui ogni elemento contiene un indice specifico. Il primo elemento dell'insieme possiede un indice di 1, il secondo elemento contiene un indice di 2, e così via. Per accedere alle voci di una raccolta, la collezione di classe è dotata di una proprietà denominata **Item**. Ci sono due modi per utilizzare questa proprietà, uno utilizzando formalmente il valore della proprietà, digitando il nome dell'oggetto collezione, seguito dall'operatore periodo e seguito da un articolo e le parentesi opzionali. Dopo la proprietà Item o all'interno delle sue parentesi, si deve digitare l'indice della voce desiderata. Ecco un esempio:

```
Sub Test()
    Dim dipendenti As Collection
    Set dipendenti = New Collection
dipendenti.Add ”Patrizia”
dipendenti.Add ”Giacomo”
dipendenti.Add ”Alice”
dipendenti.Add ”Franco”
dipendenti.Item 2
End Sub
```

Oppure si possono usare anche le parentesi:

```
Sub Test()
    Dim dipendenti As Collection
    Set dipendenti = New Collection
dipendenti.Add ”Patrizia”
dipendenti.Add ”Giacomo”
dipendenti.Add ”Alice”
dipendenti.Add ”Franco”
dipendenti.Item (2)
End Sub
```

Invece di utilizzare la proprietà Item, è possibile applicare l'indice direttamente all'oggetto collezione. Ecco alcuni esempi:

```
Sub Test()
    Dim dipendenti As Collection
    Set dipendenti = New Collection
dipendenti.Add ”Patrizia”
dipendenti.Add ”Giacomo”
dipendenti.Add ”Alice”
dipendenti.Add ”Franco”

 dipendenti.Item 2
 dipendenti.Item (2)
 dipendenti 2
 dipendenti (2)
End Sub
```

Tutte queste quattro tecniche (quelle in rosso) consentono di accedere alla voce di cui abbiamo specificato il suo indice

Rimozione di un elemento da una raccolta

In contrasto con l'aggiunta di un nuovo elemento, è possibile eliminare uno. A sostegno di questa operazione, la classe Collection è dotato di un nome metodo **Remove**. La sua sintassi è:

```
Public Sub Remove(Index As Integer)
```

Questo metodo accetta un argomento e quando si richiama, si deve passare l'indice della voce che si desidera eliminare. Ecco un esempio:

```
Sub Test()
    Dim dipendenti As Collection
    Set dipendenti = New Collection
dipendenti.Add ”Patrizia”
dipendenti.Add ”Giacomo”
dipendenti.Add ”Alice”
dipendenti.Add ”Franco”
 dipendenti.Remove 2
End Sub
```

Questo codice elimina il secondo elemento della collezione.

Numero di elementi in una Collezione

Quando si inizia una nuova collezione, ovviamente è vuota e il suo numero di elementi è 0. Per tenere traccia del numero di elementi in una collezione, la collezione di classe è dotata di una proprietà denominata **Count** il cui tipo è un numero intero. Si deve ricordare che tutte le classi di Collection incorporate ereditano il loro comportamento dalla classe Collection, ciò significa che le classi di raccolta incorporate sono dotate di una proprietà denominata Count per contenere il numero di elementi.

Abbiamo visto come è possibile aggiungere nuovi elementi a un insieme e ogni volta che si aggiunge un nuovo elemento alla raccolta, la proprietà Count aumenta di 1. Sappiamo anche come rimuovere un elemento da una raccolta e ogni volta che un elemento esistente viene eliminato, il valore della proprietà Count è diminuito di 1.
In qualsiasi momento, possiamo conoscere il numero di elementi di una collezione, ottenendo il valore della sua proprietà Count

Operatori di confronto, logici e matematici

Per combinare o confrontare specifici valori in un'espressione si usano gli **operatori**. Il loro nome deriva dal fatto che essi sono i simboli che indicano specifiche operazioni matematiche o di altro genere da eseguire su vari valori in un'espressione. Quando in un'espressione si usa un operatore, i dati, che siano variabili o costanti su cui l'operatore agisce vengono detti operandi. Nell'espressione **2+1**, per esempio, i numeri **2** e **1** sono gli operandi dell'operatore di somma (+). Adesso vediamo i vari tipi di operatori e come si usano

Operatori di confronto

Gli operatori di confronto, chiamati a volte operatori relazionali, vengono utilizzati soprattutto per stabilire il criterio, in base al quale prendere delle decisioni. Il risultato di un'operazione di confronto è sempre di tipo *Boolean*: *True* o *False* e sono usati per comparare valori letterali, costanti o variabili di ogni tipo. La tabella rappresentata in **figura 1** elenca gli operatori di confronto disponibili nel VBA

Operatore	Significato
=	Uguaglianza
>	Maggiore di
<	Minore di
>=	Maggiore o uguale
<=	Minore o uguale
<>	Diverso
In	All'interno di un insieme
Is Null	Il campo è vuoto
Is Not Null	Il campo non è vuoto
Between	Tra due valori

Fig. 1

Se entrambi gli operandi di un'espressione di confronto sono dello stesso tipo di dati, il VBA esegue il confronto diretto per quel tipo di dati, per esempio se entrambi sono stringhe, VBA compara le due stringhe, se sono date VBA compara le date e così via. Utilizzando gli operatori, è possibile impostare delle condizioni più sofisticate alle istruzioni che andremo a scrivere. Aiutiamoci con un esempio per comprendere l'utilizzo degli operatori di confronto

4 <> 5 => il risultato è True, 4 è diverso da 5
Alex > Carlo => il risultato è False, "Alex" è più corto di "Carlo"
Abc = abc => il risultato è False, Abc è diverso da abc

Possiamo dire, In maniera molto sintetica, che questo tipo di operatore altro non fa che confrontare due variabili e appurare se la condizione che chiediamo tramite il simbolo dell'operatore, sia vera o falsa e il risultato del confronto è sempre un valore di tipo Boolean. Le variabili booleane riportano o accettano solo due valori, Vero(True) e Falso(False) e possiamo impostarle in questo modo

Dim alex As Boolean
alex = True

Dalle due righe di codice abbiamo dedotto che è stata dichiarata la variabile alex con tipo di dati Boolean invece di impostarla come tipo di dati con Integer o Stringa. La seconda linea di codice inserisce un valore nella variabile, ma non è un dato che possiamo manipolare, mettiamo solo una "condizione" cioè diciamo che la variabile alex è uguale a Vero. Vediamo qualche riga di codice per comprendere meglio come usarla

```
If alex = True Then
MsgBox "E 'vero"
Else
MsgBox "E 'falso"
End If
```

La prima linea del ciclo **If** verifica se la variabile alex corrisponde a un valore Vero, se lo è, allora visualizza un messaggio, dato che ci sono solo due opzioni da testare (True e False) possiamo avere una parte del ciclo If per testare un valore (Che può essere True) e se non viene soddisfatto sarà sicuramente il contrario (False). Possiamo approfondire con il codice seguente

```
Sub operatori()
Dim alex As Integer
alex = 10
If alex < 20 Then
MsgBox alex & " " & "è inferiore a 20"
End If
End Sub
```

Se osserviamo il codice sopra riportato notiamo che abbiamo creato una variabile (alex), l'abbiamo dichiarata di tipo *Integer* (Dim alex As Integer) e memorizzato il valore 10 in essa (alex=10), ma se osservate la prima riga della dichiarazione **If**, viene posta la condizione If alex < 20 Then, se si consulta la tabella di **figura 1** vedrete che il simbolo < significa Minore di, quindi la condizione posta è "Se alex è inferiore a 20" la condizione restituisce TRUE e il codice tra If e End If viene eseguito, se invece la condizione restituisce FALSE allora VBA salterà tutte le righe di codice e passerà alla prima istruzione che trova dopo End If. Se eseguiamo la macro otteniamo una finestra come la seguente

Fig. 2

Possiamo usare altri operatori esposti in figura 1 in tutti i casi che possono aiutarci a porre delle condizioni come per esempio utilizzando un ciclo If in questo modo

```
Sub operatori()
Dim alex As Integer
alex = 19
If alex = 20 Then
MsgBox alex & " " & "è uguale a 20"
ElseIf alex > 20 Then
MsgBox alex & " " & "è maggiore di 20"
Else
MsgBox alex & " " & "è inferiore a 20"
End If
End Sub
```

Che riporta un messaggio come il seguente

Fig. 3

Per concludere l'argomento possiamo riassumere affermando che Il risultato di un operatore di confronto è un cosiddetto valore di verità: può essere vero(True) oppure falso(False)

Operatori Matematici

Il VBA è in grado di eseguire tutte le operazioni aritmetiche normali come somma, sottrazione, divisione e moltiplicazione in quanto l'elaborazione di dati numerici è una delle attività principali di un programma. VBA supporta una serie di operatori matematici che possono essere usati negli enunciati di un programma. Queste operazioni, con il relativo simbolo che identifica l'operatore matematico, sono riportate nella tabella di **figura 4**

Operatore	Significato
+	Somma
-	Sottrazione
*	Moltiplicazione
/	Divisione
\	Divisione intera
Mod	Resto della divisione
&	Concatenazione di 2 stringhe

Fig. 4

Avrete certamente notato nelle linee di codice sopra esposte che abbiamo usato un simbolo particolare dopo l'istruzione MsgBox. Abbiamo inserito la variabile alex poi uno spazio e poi una E commerciale (**&**).

Questo simbolo in VBA serve per concatenare (unire) due o più stringhe. Infatti dopo la & abbiamo uno spazio seguito da una stringa vuota posta virgolette, poi ancora una &, uno spazio e il testo del messaggio racchiuso tra virgolette. Quindi, con la & abbiamo unito la variabile e il testo separandolo da uno spazio vuoto

Se **NON** avessimo usato la concatenazione tra la variabile e il testo del messaggio avremmo ricevuto un errore di compilazione come il seguente

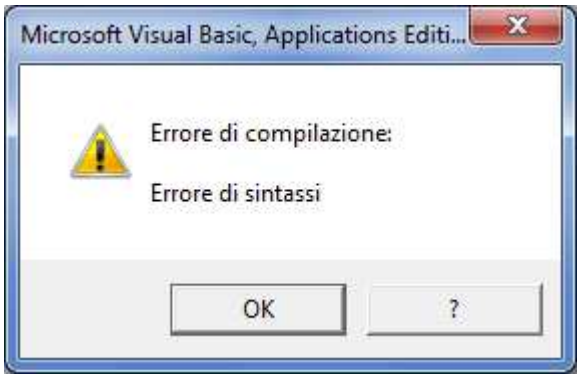


Fig. 5

Al tempo stesso concatenando la variabile col testo del messaggio senza inserire lo spazio vuoto avremmo ottenuto un box come il seguente

Fig. 6

Come si può notare il valore della variabile alex risulta essere attaccato al testo del messaggio, che sicuramente è poco leggibile e interpretabile. Per meglio comprendere possiamo usare il codice sotto riportato

```
Sub Prova()
a = "Auguro a tutti"
b = "gli utenti del Pianeta"
c = "Buon Natale e Felice Anno Nuovo" >
MsgBox a & b & c
End Sub
```

Che riporta un messaggio come il seguente

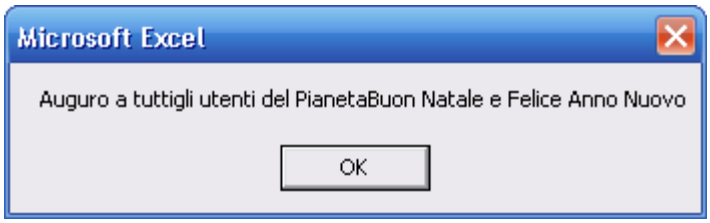


Fig. 7

Fate attenzione che se avete all'inizio delle routine la parola chiave Option Explicit vi viene rimandato un errore, in quanto come abbiamo detto poco sopra in presenza di questa parola chiave tutte la variabili vanno dichiarate.

Se osservate attentamente la **Fig. 9** noterete che ci sono delle frasi attaccate (tuttigli e PianetaBuon), modifichiamo allora il nostro codice utilizzando un *operatore di concatenazione* (&) in questo modo

```
Sub Prova1()
a = "Auguro a tutti"
b = "gli utenti del Pianeta"
c = "Buon Natale e Felice Anno Nuovo"
MsgBox a & " " & b & " " &  c
End Sub
```

Con questo ulteriore concatenamento abbiamo inserito uno spazio vuoto tra una frase e l'altra, infatti la sintassi " " (uno spazio vuoto racchiuso dalle virgolette) sta ad indicare uno spazio vuoto e di conseguenza la nostra scritta ci appare così

Fig. 8

Con questi operatori possiamo anche fare operazioni matematiche usando le variabili, se prendiamo questo codice, ricordando che le variabili Alex, x e y sono state dichiarate come *Integer* nella barra delle dichiarazioni quindi condivisibili in tutto il modulo

```
Sub somma()
Dim operator As Integer
Alex = 10
x = 20
y = 30
operator = y - x + Alex
MsgBox "Il risultato della tua operazione è" & " " & operator
End Sub
```

Questo codice equivale a: Prendi il valore della variabile **y**, sottrai il valore della variabile **x** e somma il valore della variabile *Alex* eseguendo questa macro ci verrà riportato questo avviso

Fig. 9

Operatori Logici

Il più delle volte gli operatori logici forniti dal VBA vengono usati per combinare i risultati di singole espressioni di confronto al fine di costruire criteri complessi per prendere una decisione all'interno di una procedura, oppure per stabilire le condizioni in base alle quali un enunciato o un gruppo di istruzioni possa essere replicato. Come operando di un operatore logico si può usare qualsiasi espressione valida che dia un risultato Boolean, o un numero che il VBA possa convertire in valore booleano. Il VBA considera 0 come equivalente a False e ogni altro valore numerico equivalente a True. Nella tabella di Figura 12 viene fornito l'elenco degli operatori logici presenti nel VBA

Operatore	Significato
And	Entrambe le espressioni sono vere
Or	E' vera una o l'altra espressione
Not	L'espressione non è vera

Fig. 10

Per poter comprendere al meglio la struttura degli operatori logici è importante saper leggere la tabella delle verità Booleana, essa non è altro che una tabella che mostra tutte le possibili combinazioni di valori per una particolare espressione logica e il risultato di ognuna. Questa tabella ha 3 colonne, la prima contiene il valore del primo operando, la seconda il valore del secondo operando e la terza il valore del risultato dell'espressione. Guardando la riga sotto esposta

False True False

In essa il primo operando è False, il secondo True e il risultato dell'operatore **And** con questi valori è *False*. La riga in questione, perciò, insegna che il risultato dell'espressione *False And True è False*. La sintassi dell'operatore And è la seguente:

Operando1 And Operando2

Vediamo ora la tabella della verità booleana per l'operatore And

- True True True
- True False False
- False True False
- False False False

Dove la prima colonna rappresenta il valore booleano del primo valore. La seconda il valore booleano del secondo valore e la terza colonna il risultato dell'espressione.

L'operatore **And** effettua una congiunzione logica (detta anche somma logica) e il risultato di un'operazione And è True solo se ambedue gli operatori sono True altrimenti è False. L'operatore And viene utilizzato per determinare se due diverse condizioni sono vere contemporaneamente. Per esempio:

(ricavo < 5000) And (profitto < 1000)

L'espressione precedente risulta vera (True) se il valore di ricavo è minore di 5.000 e al tempo stesso il valore del profitto è minore di 1.000 (e solo in quel caso). Si noterà che i due operandi in questa espressione sono espressioni di confronto e anche che sono poste tra parentesi per renderli operandi dell'operatore And. Le parentesi indicano al VBA di calcolare il risultato dell'espressione tra parentesi prima di valutare altre parti dell'espressione completa. Le parentesi, inoltre, rendono l'espressione più leggibile raggruppando le parti correlate di un'espressione. L'uso in questo modo delle parentesi, per raggruppare parti di un'espressione in una sotto-espressione è molto comune con espressioni di ogni tipo, numeriche, di stringa, di data e di confronto.

L'operatore Or effettua una disgiunzione logica, spesso specificata anche come or inclusivo. Il risultato di un'operazione Or è True solo se uno o entrambi gli operandi è True, altrimenti è False. L'operatore *Or* ha questa sintassi:

Operando1 Or Operando2

La tabella della verità booleana per l'operatore Or è la seguente:

- True True True
- True False True
- False True True
- False False False

Si userà l'operatore *Or* per determinare se una o l'altra di due condizioni sia vera. Per esempio

(ricavo < 5000) Or (profitto < 1000)

Il risultato sarà True se il valore di ricavo è minore di 5.000 oppure il valore di profitto è minore di 1.000

L'operatore **Not** effettua la negazione logica, cioè inverte il valore dell'unico operando. Il risultato quindi è True se l'operando è False e False se l'operando è True.

Passaggio di argomenti alle procedure

Quando un valore esterno deve essere utilizzato da una procedura per eseguire un'azione, si passa alla procedura da variabile. Queste variabili che sono passate a una procedura sono chiamate argomenti che rappresenta il valore fornito dal codice chiamante a una procedura. Nella sintassi di dichiarazione della procedura compaiono degli elementi racchiusi tra parentesi quadre: sono i parametri o argomenti e quando il set di parentesi, dopo il nome della procedura nella dichiarazione Sub o la dichiarazione Function, sono vuoti, si tratta di un caso in cui la procedura non riceve argomenti. Tuttavia, quando gli argomenti sono passati a una procedura da altre procedure, allora questi sono elencati o dichiarati tra le parentesi.

I parametri hanno un ordine posizionale, vanno elencati separati da una virgola e devono rispettare il tipo di dato specificato nella dichiarazione, inoltre possono essere "passati" come valori fissi o tramite variabili. In sostanza occorre dare un nome ad ogni parametro e specificare il tipo di dato che conterrà (come per le variabili), se ne indichiamo più di uno occorre elencarli di seguito separati da una virgola.

Tipi di argomenti di dati

Nell'esempio che segue la dichiarazione della funzione '*voto*' contiene una variabile di tipo Integer passata come argomento e restituito un valore di tipo String. E' bene ricordare che solitamente si dichiara il tipo di dati per gli argomenti passati a una procedura, nel caso venisse omessa questa dichiarazione, il tipo di dati predefinito che VBA assegna è di tipo Variant.

```
Function voto(votazione As Integer) As String
If votazione >= 80 Then
voto = "A"
ElseIf votazione >= 60 Then
voto = "B"
ElseIf votazione >= 40 Then
voto = "C"
Else
voto = "Pessimo"
End If
End Function

Sub callvoto()
Dim i As Integer
Dim str As String
i = InputBox("Inserisci il voto per Marco")
 str = voto(i)
MsgBox "La valutazione di Marco è : " & str
End Sub
```

Nella routine *callvoto* è stata chiamata la funzione *voto* e il risultato ottenuto assegnato alla variabile str variabile. Nell'esempio sottostante la dichiarazione della funzione votiPercent contiene due variabili come argomenti, con un valore Double come tipo di dati di ritorno.

```
Function votiPercent(voti As Integer, Totalvoti As Integer) As Double
votiPercent = voti / Totalvoti * 100
votiPercent = Format(votiPercent, "#.##")
End Function

Sub callvoti_1()
Dim voti As Integer
Dim Totalvoti As Integer
Dim pcnt As Double
voti = InputBox("Inserisci Voti")
Totalvoti = InputBox("Inserisci Totale Voti")
pcnt = votiPercent(voti, Totalvoti)
MsgBox "La percentuale è del :" & " " & pcnt & "%"
End Sub
```

L'ordine posizionale è anche quello che ci servirà per riferirci ad essi quando utilizziamo la procedura. E' possibile passare dei parametri alle procedure ed alle funzioni in due modi:

- *Passaggio per riferimento (ByRef)*. Si utilizza quando all'interno della procedura (o funzione) il valore del parametro viene modificato e, alla procedura chiamante, serve il valore di ritorno modificato
- *Passaggio per valore (ByVal)*. Il valore del parametro, ritorna alla procedura chiamante con il suo valore originale, anche se viene modificato all'interno della procedura (o funzione).

Passaggio di argomenti per valore

Quando si passa un argomento per valore in una procedura, solo una copia di una variabile viene passata e qualsiasi modifica del valore della variabile nella procedura corrente non influenza o modifica la variabile stessa nella sua posizione originale perché la variabile stessa non è accessibile. Per passare un argomento per valore, si deve utilizzare la parola chiave **ByVal** . Vediamolo con un esempio

```
Sub Prova_1()
Dim x As Integer
x = 5
MsgBox x
Call Aggiungi_1 (x)
MsgBox x
End Sub

Sub Aggiungi_1(ByVal i As Integer)
i = i + 1
End Sub
```

In questo codice viene assegnato il valore '5' alla variabile x che tramite la funzione MsgBox ne riporta il valore sullo schermo, successivamente viene chiamata la procedura '*Aggiungi_1*' che prende come valore in entrata una copia del valore della variabile x e lo incrementa di 1 unità.

Questo passaggio non ha effetto sul contenuto della variabile x, infatti quando viene stampato a video il valore di ritorno il valore di x è sempre 5. Modifichiamo le istruzioni in questo modo

```
Sub Prova_2()
Dim x As Integer
x = 5
MsgBox x
Call Aggiungi_2 (x)
MsgBox x
End Sub

Sub Aggiungi_2(ByRef i As Integer)
i = i + 1
End Sub
```

Questa volta la variabile è stata passata per riferimento Questo significa che la procedura va a modificare il contenuto della variabile x Infatti alla fine dell'elaborazione il valore di x è stato incrementato e vale 6. Bisogna fare attenzione che di default il VBA considera le variabili passate per riferimento e quindi ciò che si modifica in una procedura ha effetto sulla variabile che viene passata. Ognuna delle due modalità risulta vantaggiosa se applicata con una finalità opportuna, Il passaggio per valore permette di essere sicuri che la procedura non modificherà le variabili contenute nel programma che sta usando tali procedure, mentre Il passaggio per riferimento è più veloce perché non deve eseguire una copia. Inoltre il passaggio per riferimento permette di agire su un numero arbitrario di variabili

Mentre una funzione può restituire un unico valore al termine della sua esecuzione, una procedura può prendere in ingresso per riferimento più variabili e modificarne tutti i valori, tuttavia l'uso delle funzioni permette di utilizzare il nome della funzione stessa all'interno di espressioni più complesse

```
Function calcola_comm(ByVal comm_tass As Double, ByVal vendite As Currency) As
Currency
calcola_comm = comm_tass * vendite
End Function

Sub vendite_marco()
Dim comm_marco As Currency
Dim comm_tass As Double
Dim vendite As Currency
comm_tass = InputBox("Inserisci tasso commissione")
vendite = InputBox("Inserisci Importo Vendite")
comm_marco = calcola_comm(comm_tass, vendite)
MsgBox "La commissione è di" & " " & comm_marco
End Sub
```

Passaggio di argomenti per riferimento

Quando si passa un argomento per riferimento in una procedura, la variabile stessa accede alla procedura e il valore della variabile viene modificato in modo permanente dalla procedura stessa. Per passare un argomento per riferimento, si deve utilizzare la parola chiave **ByRef** prima l'argomento da passare, inoltre il passaggio di argomenti per riferimento è anche l'impostazione predefinita in VBA, a meno che non sia esplicitamente specificato di passare un argomento per valore.

```
Function numero(ByVal i As Integer) As Long
i = 5
numero = i
End Function

Sub prova_numero()
Dim n As Integer
MsgBox numero(n)
MsgBox n
End Sub
```

In questo esempio la variabile numero restituisce il valore di 5 in quanto è la funzione numero che assegna un valore alla variabile n, ma poiché la variabile è stata passata per valore nella funzione, qualsiasi modifica dello stesso rimane nella funzione corrente e quando la funzione termina, il valore della variabile n ritornerà al valore di quando è stata dichiarata, che era pari a 0. Infatti con il secondo messaggio viene riportato il valore 0. Se la variabile fosse stata passata per riferimento nella funzione, la variabile n avrebbe definitivamente assunto il nuovo valore assegnato.

In questo esempio invece si vede il passaggio di un argomento per riferimento in una procedura utilizzando la parola chiave **ByRef**

```
Function numero_1(ByRef i As Integer) As Long
i = 5
numero_1 = i
End Function

Sub prova_numero_1()
Dim n As Integer
MsgBox numero_1(n)
MsgBox n
End Sub
```

Il valore della variabile n è impostato a 0 quando viene dichiarata e il messaggio restituito è 5 perché il richiamo della funzione numero_1, assegna un valore alla variabile n (5) e il messaggio successivo restituisce ancora 5, perché la variabile è stata passata per riferimento nella funzione numero_1, e ha definitivamente assunto il nuovo valore assegnato dalla funzione numero_1

Argomenti facoltativi

Gli argomenti possono essere specificati come facoltativi, utilizzando la parola chiave *Optional* prima dell'argomento e quando si specifica un argomento con optional, tutti gli argomenti seguenti devono essere specificati come Optional. Si noti che specificando la parola chiave Optional rende un argomento opzionale altrimenti sarà richiesto l'argomento.

L'argomento opzionale dovrebbe essere, anche se non necessario, dichiarato come tipo di dati *Variant* per consentire l'uso della funzione **IsMissing** che funziona solo quando viene utilizzato con le variabili dichiarate come Variant. La funzione IsMissing viene utilizzata per determinare se l'argomento opzionale è stata accettata nella procedura o meno, in modo da regolarsi di conseguenza nel codice senza restituire un errore. Se l'argomento opzionale non è dichiarato come Variant, la funzione IsMissing non funziona, e all'argomento opzionale verrà assegnato e il valore predefinito per il tipo di dati che è 0 per le variabili di tipo numerico (cioè Integer, Double, etc.) e Nothing per String o variabili di tipo Object.

Esempio: Dichiarazione della routine con due argomenti Optional

```
Sub dip_nome1(Optional primo As String, Optional secondo As String)
ActiveSheet.Range("A1") = primo
ActiveSheet.Range("B1") = secondo
End Sub

Sub nome_pieno1()
Dim nome1 As String
Dim nome2 As String
nome1 = InputBox("Inserisci il primo Nome")
nome2 = InputBox("Inserisci il secondo Nome")
Call dip_nome1(nome1, nome2)
End Sub
```

Esempio: Dichiarare l'argomento Optional come String, senza utilizzare la funzione IsMissing.
La dichiarazione della routine contiene due argomenti, il secondo argomento è specificato come Optional. Si noti in questo esempio che l'argomento opzionale viene dichiarato come String e non è passato nella procedura, di conseguenza il Range ("B1"), conterrà un riferimento *Null* dopo aver eseguito la sub *dip_nome2* perché all'argomento opzionale verrà assegnato il valore di default per il suo tipo di dati (String), che è Nothing cioè un riferimento Null.

Se l'argomento opzionale viene dichiarato come Integer, Range ("B1") conterrà zero dopo l'esecuzione della sub dip_nome2 in quanto all'argomento opzionale verrà assegnato il valore predefinito per il suo tipo di dati (Integer) che è pari a zero.

```
Sub dip_nome2(Optional primo As String, Optional secondo As String)
ActiveSheet.Range("A1") = primo
ActiveSheet.Range("B1") = secondo
End Sub

Sub nome_pieno2()
Dim nome1 As String
nome1 = InputBox("Inserisci il primo Nome")
Call dip_nome2(nome1)
End Sub
```

Esempio: Dichiarare l'argomento Optional come Variant, e utilizzare la funzione IsMissing.
La dichiarazione della routine contiene due argomenti, il secondo argomento è specificato come Optional. L'argomento opzionale dovrebbe essere (anche se non necessario), dichiarato come tipo di dati Variant per consentire l'uso della funzione IsMissing.

La funzione *IsMissing* viene utilizzata per determinare se l'argomento opzionale è stato approvato nella procedura o meno

```
Sub dividi1(primo_n As Integer, Optional secondo_n As Variant)
Dim result As Double
If IsMissing(secondo_n) Then
result = primo_n
Else
result = primo_n / secondo_n
End If
result = Format(result, "#.##")
MsgBox result
End Sub

Sub chiama_dividi1()
Dim numero1 As Integer
numero1 = InputBox("Inserire il primo Numero")
Call dividi1(numero1)
End Sub
```

Esempio: Dichiarare l'argomento Optional come Integer, ma la funzione IsMissing non funzionerà e il codice restituirà un errore.

La dichiarazione della routine contiene due argomenti, il secondo argomento è specificato come Optional. L'argomento opzionale viene dichiarato come Integer, ma la funzione *IsMissing* non funzionerà con una variabile non dichiarata come Variant. Quindi all'argomento opzionale verrà assegnato il valore predefinito per il tipo di dati che è 0 per i tipi di dati numerici e il codice restituirà l'errore: Errore di run-time "11": Divisione per zero.

```
Sub dividi2(primo_n As Integer, Optional secondo_n As Integer)
Dim result As Double
If IsMissing(secondo_n) Then
result = primo_n
Else
result = primo_n / secondo_n
End If
result = Format(result, "#.##")
MsgBox result
End Sub

Sub chiama_dividi2()
Dim numero_1 As Integer
numero_1 = InputBox("Inserisci il primo Numero")
Call dividi2(numero_1)
End Sub
```

Esempio: Dichiarare l'argomento Optional come String, ma la funzione IsMissing non funzionerà e il codice restituirà un errore.

La dichiarazione della routine contiene due argomenti, il secondo argomento è specificato come Optional. L'argomento opzionale viene dichiarato come stringa, ma la funzione IsMissing non funzionerà con una variabile non dichiarata come tipo di dati Variant. Quindi all'argomento opzionale verrà assegnato il valore predefinito per il tipo di dati che è Nothing (un riferimento Null) per il tipo di dati String e il codice restituirà l'errore: Errore di run-time "13": Tipo non corrispondente.

```
Sub dividi3(primo_n As Integer, Optional secondo_n As String)
Dim result As Double
If IsMissing(secondo_n) Then
result = primo_n
Else
result = primo_n / secondo_n
End If
result = Format(result, "#.##")
MsgBox result
End Sub

Sub chiama_dividi3()
Dim numero1 As Integer
numero1 = InputBox("Inserisci il primo Numero")
Call dividi3(numero1)
End Sub
```

Specificare un valore predefinito per un argomento facoltativo

È possibile specificare un valore predefinito per un argomento opzionale che sarà utilizzato se l'argomento opzionale non viene passato alla procedura. In questo modo è possibile dichiarare gli argomenti opzionali di qualsiasi tipo di dati e specificare un valore predefinito che sarà utilizzato se l'argomento opzionale viene omesso, evitando l'uso della funzione IsMissing che funziona solo con il tipo di dati Variant.

Esempio: Un argomento opzionale non viene passato alla procedura a viene utilizzato un valore predefinito.

La dichiarazione della routine contiene due argomenti, il secondo argomento è specificato come Optional. Se il secondo argomento, che è opzionale, non è passato nella procedura un valore predefinito viene utilizzato:

```
Sub dividi4(primoN As Integer, Optional secondoN As Integer = 3)
Dim result As Double
result = primoN / secondoN
result = Format(result, "#.##")
MsgBox result
End Sub

Sub calcula_2()
Dim numero1 As Integer
numero1 = InputBox("Inserisci il primo Numero")
Call dividi4(numero1)
End Sub
```

Passare un numero indefinito di argomenti - parametro Array (ParamArray)

Abbiamo visto come dichiarare procedure passando argomenti, tra cui argomenti opzionali, ma comunque sempre limitati al numero di argomenti dichiarati nella procedura. Usando la parola chiave **ParamArray** sarà consentito passare un numero arbitrario di argomenti alla procedura in modo che vengano accettati un numero indefinito di argomenti. Utilizzare *ParamArray* quando non si è sicuri del numero preciso di argomenti da passare a una procedura, e potrebbe anche essere conveniente creare un array opzionale (cioè un ParamArray) che passare attraverso il fastidio di dichiarare un gran numero di argomenti opzionali, e quindi utilizzando la funzione IsMissing con ciascuno di essi.

Una procedura utilizza le informazioni sotto forma di variabili, costanti ed espressioni per effettuare azioni ogni volta che viene chiamata e tramite la dichiarazione del procedimento si definisce un parametro che consente al codice chiamante (il codice che chiama la procedura) di passare un argomento, o il valore di tale parametro, in modo che ogni volta che la routine viene chiamata il codice chiamante può passare un argomento diverso per lo stesso parametro. Un parametro viene dichiarato come una variabile specificando il nome e il tipo di dati. Dichiarando una matrice di parametri, la procedura può accettare una matrice di valori per un parametro. Una matrice di parametri è così definita utilizzando la parola chiave ParamArray.

ParamArray può essere definita in una procedura ed è sempre l'ultimo parametro nell'elenco dei parametri. Un ParamArray è un parametro opzionale e può essere l'unico parametro facoltativo in una procedura e deve essere dichiarato come un array di tipo Variant. Indipendentemente dall'impostazione Option Base per il modulo, LBound di un ParamArray sarà sempre 0 e il valore indice per l'array partirà da 0. *ByVal*, *ByRef* o *keywords* sono opzionali e non possono essere utilizzate con ParamArray.

Per accedere al valore di una matrice di parametri si utilizza la funzione *UBound* per determinare la lunghezza dell'array che vi darà il numero di elementi o valori di indice nella matrice. Nel codice della procedura si può accedere al valore di una matrice di parametri digitando il nome dell'array seguito da un valore di indice (che dovrebbe essere compreso tra 0 e il valore UBound)

Esempio: Passare un numero arbitrario di argomenti utilizzando un parametro ParamArray.

```
Sub aggiungiN(ParamArray numeri() As Variant)
Dim somma As Long
Dim i As Long
For i = LBound(numeri) To UBound(numeri)
somma = somma + numeri(i)
Next i
MsgBox somma

End Sub
Sub chiama_aggiungiN()
Call aggiungiN(22, 25, 30, 40, 55)
End Sub
```

Esempio: Un argomento obbligatorio e un numero arbitrario di argomenti utilizzando ParamArray.
Una routine con un solo argomento richiesto (comm) e quindi permette di passare un numero arbitrario di argomenti utilizzando il parametro ParamArray.

```
Sub calcComm(comm As Double, ParamArray venD() As Variant)
Dim somma As Double
Dim n As Variant
For Each n In venD
MsgBox n
totalComm = totalComm + comm * n
Next n
MsgBox totalComm
End Sub

Sub chiama_Comm()
Dim c As Double
c = 0.3
Call calcComm(c, 100, 200, 300)
End Sub
```

Esempio: Due argomenti richiesti e un numero arbitrario di argomenti utilizzando ParamArray.
Una dichiarazione di routine con due argomenti richiesti (studente e media) e quindi permette di passare un numero arbitrario di argomenti utilizzando il parametro ParamArray.

```
Sub media_V(studente As String, media As Double, ParamArray voti() As Variant)
Dim n As Variant
Dim voto As String
Dim T_voti As String
For Each n In voti
If n >= 80 Then
voto = "Eccellente"
ElseIf n >= 60 Then
voto = "Buono"
ElseIf n >= 40 Then
voto = "Medio"
Else
voto = "Bocciato"
End If
If Len(T_voti) = 0 Then
T_voti = voto
Else
T_voti = T_voti & ", " & voto
End If
Next n
For Each n In voti
i = i + 1
somma = somma + n
media = somma / i
Next n
T_voti = """" & T_voti & """"
media = Format(media, "#.##")
MsgBox studente & " " & "ha voti di tipo " & T_voti & " " & "e una media voti di" & " " &
media
End Sub
```

Range, Cells e ciclo With

Abbiamo visto come è strutturato un foglio di calcolo o cartella, abbiamo detto che è costituito da celle, intervalli e fogli, inoltre un insieme di celle è rappresentato da righe e colonne, ricordando che la maggior parte dei compiti che svolgiamo in un foglio di calcolo è quella di introdurre informazioni, tagliare e copiare dati o applicare opzioni di formattazione e tante altre funzioni che coinvolgono celle, righe o colonne, questo insieme è definito **Range** . Un Range può essere rappresentato da una singola cella o più celle da una colonna, una riga o una selezione di celle, il sistema più facile per identificarlo è proprio il comando Range che ha questa sintassi

Object.Range(nome)

- ***Object**:* è un riferimento all'oggetto *Worksheet* che contiene il Range, se viene omesso VBA assume che si riferisca all'ActiveSheets (il foglio che abbiamo attivo)
- ***Nome***: è un riferimento al Range o il nome del Range inserito come testo, infatti questo comando lavora anche con Range che hanno un nome, vediamo qualche esempio

Worksheets("Foglio1").Range("A1").Value = 123, Oppure possiamo rappresentarlo come un insieme di celle in questo modo: *Worksheets("Foglio1").Range("A1:C5").Value = 123*

Possiamo usarlo anche con un nome assegnato agendo in questo modo : Selezioniamo un insieme di celle e dal Menu **Inserisci - Nome - Definisci**, come da immagine sotto

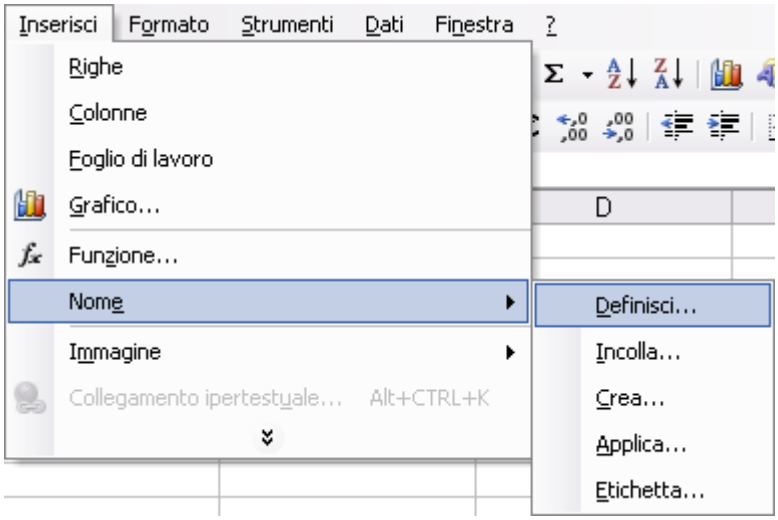


Fig. 1

E ci comparirà un box come il seguente

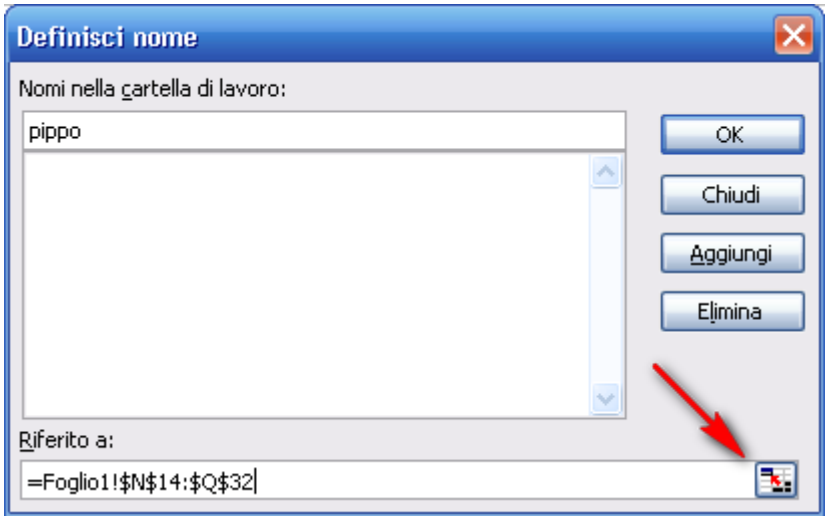


Fig. 2

NB **Per versioni di Excel superiori alla 2003** *per assegnare un nome ad un intervallo di celle si deve seguire il percorso* ***Formule - Definisci nome*** *come da immagine sotto stante*

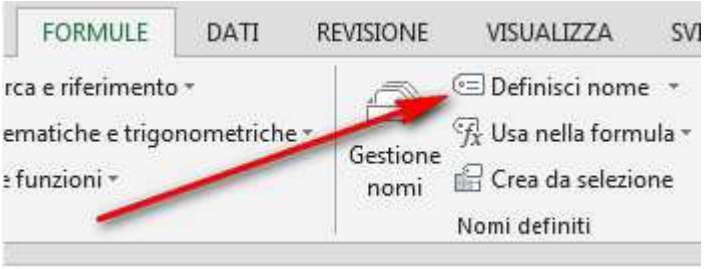


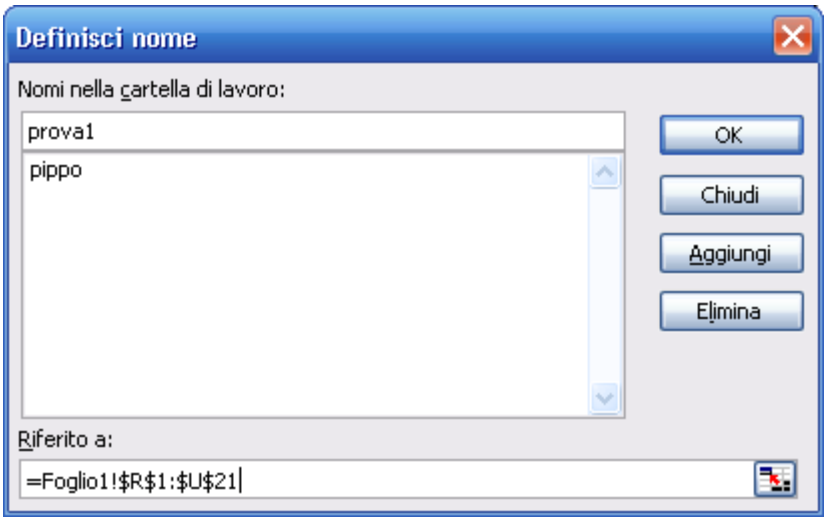
Fig. 3

E ci compare una finestra come la seguente

Fig. 4

Fine Nota

Se abbiamo selezionato prima le nostre celle troveremo il loro riferimento nel campo *Riferito a* oppure possiamo cliccare sull'icona evidenziata dalla freccia rossa e procedere alla loro selezione, fatto questo dobbiamo solo inserire il nome dell'intervallo e cliccando su *Aggiungi* lo stesso comparirà nel box centrale così:



A questo punto possiamo modificare il nostro listato in questo modo:
Worksheets("Foglio1").Range("pippo").Value = 123

Possiamo riepilogare che con il primo codice abbiamo riempito la cella A1 col valore 123, mentre col secondo abbiamo riempito un *insieme di celle* (dalla A1 alla C5) col valore 123, mentre assegnando un *nome ad un intervallo di celle* abbiamo riempito tutto l'intervallo col valore 123. Possiamo però dire anche che se siamo certi di operare nel foglio attivo posso omettere il riferimento *Object* e quanto abbiamo finora visto lo possiamo scrivere anche in questo modo

- *Range("A1").Value = 123* (per la sola cella A1)
- *Range("A1:C5").Value = 123* (per un intervallo di celle)
- *Range("pippo").Value = 123* (utilizzando un nome di un intervallo di celle)

Sempre sulla falsa riga di quanto appena citato possiamo ulteriormente semplificare il listato usando <u>riferimenti assoluti</u> in questo modo

- *[A1] = 123* (per la sola cella A1)
- *[A1:C5] = 123* (per un intervallo di celle)
- *[pippo] = 123* (utilizzando un nome di un intervallo di celle)

Come abbiamo potuto vedere il comando Range è estremamente flessibile e lo useremo spesso per poter interagire con Excel da VBA ora vediamo come comportarci quando dobbiamo leggere in un foglio e scrivere in un altro. Molte volte abbiamo la necessità di scrivere in un foglio i nostri dati e poterli salvare in un altro foglio per successive consultazioni, useremo sempre l'enunciato Range, ma gli abbineremo anche altre funzioni come *Cells* e il *ciclo With* esponiamo ora brevemente la sintassi e l'utilizzo di questi due comandi

L'enunciato Cells

Anche se possiamo usare il comando Range per riattivare una singola cella il comando **Cells** esegue lo stesso compito ma con maggior flessibilità, quando dobbiamo scrivere o leggere dati in un foglio di calcolo direttamente da VBA con l'enunciato Range siamo sempre vincolati ad una locazione ben precisa che abbiamo appena visto e denominata dal riferimento di cella, ma non sempre sappiamo dove dobbiamo leggere e scrivere, in sostanza è abbastanza difficile usare il comando Range quando dobbiamo copiare un insieme di dati scritti in varie celle estese su righe o colonne. A questo problema si può ovviare usando il comando Cells, che ha questa sintassi

Object.Cells(riga,colonna)

Per quanto riguarda il comando *Object* tralasciamo ulteriori spiegazioni in quanto vale quanto sopra esposto per il comando Range , noterete però che il riferimento al Range (nome) è espresso in coordinate di riga e colonna, questo ci permette di identificare una singola cella o un intervallo dalla loro posizione di riga e colonna, facciamo qualche esempio di identificazione di celle.

A1 = Cells (1,1)
B5 = Cells (5,2)
D3 = Cells (3,4)

Vediamo ora il ciclo With e poi sintetizziamo il tutto e uniamo i vari comandi

Il Ciclo With

VBA ci fornisce questa speciale struttura **With .... End With** che ci permette di fare riferimento alle proprietà o metodi che appartengono allo stesso oggetto senza dover specificare ogni volta il riferimento completo all'oggetto, bella come esposizione tecnica ma poco chiara vero? Esponiamo la sintassi del ciclo With e poi semplifichiamo il concetto con degli esempi, la sintassi è :

With Oggetto
Istruzioni
End With

Chiarifichiamo ora il tutto, all'inizio del corso abbiamo parlato di Metodi e Proprietà e anche di Oggetti, abbiamo esposto l'oggetto *Workbook* (che è la cartella di lavoro cioè il nostro file), l'oggetto *Worksheet* (che è il foglio di lavoro : Foglio1, Foglio2 ecc...) e l'oggetto *Range* (intervallo di celle, A1: B12, C1:D12, ecc...) nella definizione del ciclo *With* abbiamo detto che ci permette di omettere il riferimento completo all'oggetto (Workbook, Worksheet) quando le proprietà o i metodi che usiamo si riferiscono allo stesso oggetto, infatti basta dichiararlo una sola volta all'inizio della procedura With (vedi sintassi) così la nostra procedura risparmia il tempo che necessita per risolvere il riferimento all'oggetto per ogni proprietà o metodo all'interno dell'istruzione With. Vediamo un esempio e capirete subito come funziona, prendiamo come esempio i dati presenti in un foglio come in figura

	A	B
1		pippo
2		pluto
3		paperino
4		minni
5		topolino

Fig. 6

Supponiamo di trovarci nel Foglio1 e vogliamo scrivere i nostri dati nel Foglio2, per compiere questa operazione senza l'utilizzo del ciclo With dobbiamo utilizzare un listato del genere:
Worksheets("Foglio2").Range("B1").Value = Worksheets("Foglio1").Range("B1").Value
Oppure come abbiamo visto poco sopra in questo modo

Worksheets("Foglio2").Range("B1").Value = [B1].Value
Sheets("Foglio2").[B1].Value = [B1].Value

Ma così copiamo una singola cella, dobbiamo utilizzare un ciclo come abbiamo già visto in questo modo

```
Sub scrivi()
riga = 1
Do Until Sheets("Foglio1").Cells(riga, 2) = Empty
Sheets("Foglio2").Cells(riga, 2).Value = Sheets("Foglio1").Cells(riga, 2).Value
 riga = riga + 1
Loop
End Sub
```

Dobbiamo utilizzare una sintassi del genere in quanto dobbiamo incrementare il nostro contatore per poter scorrere tutti i dati presenti nel foglio di origine (nel nostro caso il Foglio1) e al tempo stesso incrementare la riga del foglio di destinazione, comunque già in questo listato abbiamo potuto vedere l'utilizzo del comando *Cells*, credo che sia chiaro come va utilizzato e a cosa serve, in ogni caso possiamo dire che con il comando Cells identifichiamo una cella ben precisa, il nostro problema sta solo nel fatto che dobbiamo dichiarare sempre il riferimento completo sia del foglio di origine che del foglio di destinazione (in presenza di un ciclo) però possiamo ovviare a tutto questo utilizzando il ciclo With in questo modo.

```
Sub scrivi_with()
j = 1
Do Until Sheets("Foglio1").Cells(j, 2) = Empty
 With Sheets("Foglio2")
 .Cells(j, 2) = Sheets("Foglio1").Cells(j, 2).Value
End With
    j = j + 1
Loop
End Sub
```

Apparentemente sembrano uguali, ma col ciclo With abbiamo evitato la dichiarazione dell'oggetto nel ciclo *Do Loop*, o meglio lo abbiamo fatto una sola volta con notevole risparmio in termini di ricerca dell'oggetto da parte di VBA e semplificando il listato in base a quanto finora citato relativo agli oggetti possiamo anche scriverlo così

```
Sub scrivi_with()
j = 1
Do Until Cells(j, 2) = Empty
 With Sheets("Foglio2")
 .Cells(j, 2) = Cells(j, 2).Value
End With
    j = j + 1
Loop
End Sub
```

Scrittura di nuove macro e procedure

Se esaminiamo una macro ottenuta con il *Registratore di macro* si potrà notare che hanno tutte alcune caratteristiche in comune. Per esempio registriamo due macro, una che formatti una selezione di celle (A1:D10) e applichi al carattere il formato grassetto e l'altra che applichi il colore rosso allo stesso intervallo di celle. Operando dal Registratore di macro, come abbiamo già visto, per la prima macro, quella che formatta le celle in grassetto otteniamo il seguente listato.

```
1) Sub Grassetto()
2)      '
3)      ' Macro Grassetto
4)      ' Macro registrata il ……
5)      '
6)      Range("A1:D10").Select
7)      Selection.Font.Bold = False
8)      Range("A1").Select
9) End Sub
```

E per la seconda macro che applica il colore rosso al carattere otteniamo il seguente listato

```
1) Sub Colore()
2)      '
3)      ' colore Macro
4)      ' Macro registrata il …….
5)      '
6)      Range("A1:D10").Select
7)      Selection.Font.ColorIndex = 3
8)      Range("A1").Select
9) End Sub
```

*Il codice sorgente di una macro in un modulo **NON** include un numero davanti ad ogni linea, La numerazione delle linee dei listati rappresentati è stata usata solo per facilitare l'identificazione e la discussione di particolari linee nei vari listati*

Come abbiamo esordito all'inizio affermando che tutte le macro hanno caratteristiche in comune, con questi due listati possiamo iniziare a notare qualche affinità. La *riga 1* è sempre l'inizio della macro e ogni macro VBA inizia con il termine ***Sub*** seguito dal nome della macro. La linea che contiene la parola chiave Sub e il nome della macro viene detta "*dichiarazione della macro*", in quanto è quella che rende noto al VBA l'esistenza della macro stessa. Il nome della macro, a sua volta, è sempre seguito da una coppia di *parentesi vuote()*, il cui scopo e utilizzo verrà affrontato più avanti nel corso, per il momento basta solo sapere che le due parentesi devono sempre comparire dopo il nome della macro nel codice sorgente. Un'altra affinità è rappresentata dalla parola chiave ***End Sub*** alla *riga 9* che segnala al VBA che è stata raggiunta la fine della macro.

Dalla *riga 2* alla *riga 5* dei due listati sono presenti dei commenti. Un commento è una riga della macro che non contiene istruzioni ma che fanno parte della macro stessa, generalmente viene usato per dei chiarimenti sull'azione che sta per compiere la macro stessa. Come si può notare, ogni riga di commento inizia con un apostrofo ('') e il VBA tratta ogni riga che comincia con un apostrofo come un commenta e non processa il testo al suo interno. Subito dopo la dichiarazione *Sub nome_della_macro ()* segue il corpo della macro che può includere varie righe di codice e/o di commento.

Nel caso della macro *Grassetto* le righe da 6 a 8 costituiscono il corpo della macro, così come la macro *Colore*. Ogni riga del corpo della macro consiste in uno o più enunciati che rappresentano una serie di parole chiave e altri simboli che assieme costituiscono un'istruzione VBA completa.

Gli enunciati VBA di una macro registrata contengono le istruzioni che svolgono azioni equivalenti a quelle che sono state eseguite quando si è registrata la macro stessa. Per esempio la riga 6 della macro *Grassetto* seleziona il Range di celle A1:D10 così come la stessa riga della macro *Colore*, mentre invece la riga 7 della macro *Grassetto* fissa il carattere al formato grassetto, la stessa riga della macro *Colore* fissa il colore del testo a rosso. La riga 8 dei due listati seleziona la cella A1 in quanto dopo aver formattato le celle abbiamo cliccato sulla cella A1.

Quando si esegue una macro il VBA parte dalla prima riga del corpo della macro (la prima dopo la dichiarazione) eseguendo le istruzioni contenute in essa in sequenza, da sinistra verso destra, poi passa alla riga successiva ed esegue le istruzioni in essa contenute e così di seguito fino a raggiungere la fine della macro, definita dall'enunciato *End Sub*. Possiamo inoltre notare che alcune parti del testo della macro sono visualizzate in colori differenti, i commenti sono visualizzati in colore verde, mentre le parole chiave *Sub* e *End Sub* e altre del VBA sono di colore blu, mentre il testo rimanente della macro è di colore nero, a indicare che si tratta di dati e enunciati creati dall'utente. Il colore con cui VBA contraddistingue le varie parti di testo sullo schermo servono a rendere più chiara la parte della macro o di un enunciato che si sta esaminando.

Modifica del testo di una macro

Per apportare delle modifiche al codice sorgente di una macro in un modulo si usano comandi e tecniche che risultano già famigliari a chi usa Windows, la modifica del testo in un modulo visualizzato nella finestra Codice avviene esattamente come quando si edita un testo in Word o in qualsiasi altro elaboratore di testi, per aggiungere, eliminare, selezionare, tagliare etc. si useranno quindi i medesimi comandi da tastiera o col mouse. Le modifiche apportate a un modulo si possono salvare tramite il comando **File - Salva** dell'Editor di VB oppure facendo clic sul pulsante *Salva* della barra degli strumenti. Ogni modifica effettuata su un modulo viene comunque salvata quando si salva la cartella di lavoro che contiene il modulo stesso.

Per fare un esempio di come effettuare modifiche a una macro prendiamo in esame la prima macro registrata (Grassetto) e andiamo a modificarla, oltre ad applicare il formato grassetto vogliamo anche cambiare dimensioni al carattere. Possiamo procedere in questo modo, portiamoci nell'Editor di VBA nella macro Grassetto e spostiamo il punto di inserimento (il cursore) alla fine della riga 4 e premiamo Invio per inserire una riga vuota. Digitiamo un apostrofo ('') e digitiamo poi il testo del commento (per esempio: la linea successiva cambia la dimensione del carattere) a questo punto nella riga successiva al commento dobbiamo solo scrivere il codice VBA che esegua l'operazione richiesta.

Possiamo aiutarci col registratore di macro, quando abbiamo trattato il registratore abbiamo detto che una volta attivato **registra tutte le operazioni svolte dall'operatore** e le converte in codice VBA, pertanto possiamo ritornare a Excel, attivare la registrazione di una nuova macro, selezionare l'intervallo di celle riportato nella macro Grassetto e modificare la dimensione del carattere. Una volta eseguita questa operazione fermiamo la registrazione e torniamo nell'Editor VBA per vedere il codice, che si presenta in questo modo:

```
Sub Macro2()
'
' Macro2 Macro
' Macro registrata il ……
'
    Range("A1:D10").Select
    With Selection.Font
        .Name = "Arial"
        .Size = 12
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = xlAutomatic
    End With
End Sub
```

Possiamo adesso prelevare il codice che ci interessa e aggiungerlo alla macro *Grassetto* in modo che compia tutte e due le operazioni, cioè che applichi il formato grassetto e inserisca il carattere Arial da 12 Pt. Possiamo ridurre il listato riportato a queste istruzioni

```
With Selection.Font
        .Name = "Arial"
        .Size = 12
End With
```

Che sono quelle che si occupano di modificare la dimensione del carattere, vedremo più avanti il significato delle altre istruzioni quando tratteremo il *Ciclo With*, per il momento consideriamo la sola modifica di una macro utilizzando il registratore di macro quando non si conosce il codice VBA da inserire. A questo punto possiamo prelevare il codice riportato e aggiungerlo alla macro Grassetto semplicemente selezionandolo e copiandolo premendo consecutivamente i tasti **Ctrl+C** oppure passando dal menu **Modifica - Copia**.

A questo punto andiamo nella macro Grassetto e posizioniamo il cursore sulla riga sotto al commento che abbiamo inserito in cui abbiamo scritto che veniva modificata la dimensione del carattere e premiamo consecutivamente i tasti **Ctrl+V** oppure dal menu **Modifica - Incolla**.
Adesso la macro è diventata

```
Sub Grassetto()
'
' Grassetto Macro
' Macro registrata il ……
' La linea successiva cambia la dimensione del carattere
Range("A1:D10").Select
      With Selection.Font
      .Name = "Arial"
      .Size = 12
       End With
Selection.Font.Bold = False
Range("A1").Select
End Sub
```

A questo punto la macro ***Grassetto*** è stata modificata usando il registratore di macro per trasformare le nostre azioni compiute sul foglio in codice sorgente e se eseguiamo la macro adesso oltre ad applicare il formato grassetto applica anche il carattere Arial da 12 Pt. Si consiglia di aggiungere commenti al corpo della macro che spiegano i motivi e lo scopo delle modifiche apportate a una macro che potranno facilitare l'identificazione delle parti revisionate e far capire meglio che cosa fanno le modifiche

Scrittura di nuove macro e procedure

Per scrivere una propria macro senza ricorrere al registratore di macro si può digitare la macro entro un modulo esistente o creare un nuovo modulo, finora abbiamo utilizzato il termine macro per identificare sia quelle scritte usando il registratore di macro che quelle "*autonome*" cioè scritte di proprio pugno, esiste però un termine che aiuta a distinguere tra macro registrate e "autonome" e, strettamente parlando, il termine macro dovrebbe essere applicato solo alle macro registrate, mentre invece le macro scritte partendo da zero sono chiamate *procedure sub* (subroutine) o semplicemente *procedure*, pertanto nel proseguo di questo corso verrà utilizzato il termine macro per identificare il codice sorgente ottenuto usando il registratore di macro e procedure per riferirsi al codice VBA scritto dall'utente

Per scrivere una nuova procedura entro un modulo esistente bisogna prima aprire la finestra Codice facendo doppio clic sul modulo in Gestione progetti oppure selezionando il modulo e poi scegliere **Visualizza - Codice**. Per scrivere il testo del codice di una procedura, che venga aggiunta ad un modulo nuovo o a uno esistente, si deve portare il cursore di testo nella finestra Codice sul punto del modulo in cui si vuole digitare la nuova procedura. Si può battere il testo di una nuova procedura in una posizione qualsiasi di un modulo, purchè essa venga comunque posizionata dopo l'enunciato End Sub che segna la fine della procedura precedente e prima dell'enunciato Sub della procedura immediatamente successiva, personalmente consiglio di aggiungere una nuova procedura sempre alla fine del modulo.

Quando si scrive una procedura bisogna specificarne il nome e includere la parola chiave *Sub* e *End Sub* rispettivamente all'inizio e alla fine della procedura stessa, se si trascura uno di questi tre particolari la sintassi della procedura non risulterà corretta e l'Editor di VB visualizzerà un messaggio di errore quando si eseguirà la procedura. Si deve tenere presente che se digitiamo il nome della procedura preceduto dalla parola chiave Sub e battiamo il tasto invio l'Editor completerà la procedura aggiungendo le due parentesi vuote e la parola chiave End Sub

Il primo classico programma che si scrive in un qualsiasi linguaggio di programmazione, per tradizione, è quello che visualizza a video la frase: *Ciao Mondo!* Il listato che segue mostra un programma VBA con una singola procedura che adempie allo scopo.

```
Sub Mondo()
MsgBox "Ciao Mondo!"
End Sub
```

Per realizzare questo listato ecco come procedere.

- Aprite una cartella di lavoro di Excel o create una nuova cartella
- Premete i tasti **Alt+F11** per entrare nell'Editor VB
- In *Gestione progetti* selezionate il documento o la cartella in cui si vuole memorizzare il programma
- Scegliere dal menu **Inserisci - Modulo** per aggiungere un nuovo modulo al progetto. L'Editor VB aggiunge un nuovo modulo e apre la finestra Codice
- Cambiate il nome del modulo, risulta più intuitivo riconoscere il contenuto del modulo dal nome, per esempio mettete saluto
- Essendo nuovo il modulo iniziate a scrivere dalla prima riga, in caso contrario accertatevi di scrivere il testo all'inizio di una riga e digitate il listato che abbiamo visto poco sopra premendo invio al termine di ogni riga

L'Editor VB include varie funzioni che aiutano a scrivere le procedure, in primo luogo non appena si preme invio dopo aver digitato la parola chiave Sub seguita dal nome della procedura, automaticamente viene pure aggiunta la riga con la parola chiave End Sub, in tal modo non occorre preoccuparsi di una possibile dimenticanza di questo elemento critico della procedura. Inoltre l'Editor VB comprende una funzionalità detta *informazioni rapide automatiche* e non appena si digita la parola chiave *MsgBox* e si preme la *barra spazio* compare una finestrella rettangolare che mostra l'elenco completo di tutti gli argomenti della procedura o funzione VBA integrata in questione, in questo caso gli argomenti di MsgBox

```
msgbox
MsgBox(Prompt, [Buttons As VbMsgBoxStyle = vbOKOnly], [Title], [HelpFile], [Context]) As VbMsgBoxResult
```

Fig. 1

L'argomento che si prevede sia il prossimo ad essere digitato compare in grassetto in questa finestrella (un argomento è un'informazione richiesta dalla procedura per poter svolgere il suo lavoro) la finestrella Informazioni rapide automatiche si chiude non appena si preme Invio per iniziare una nuova riga nella finestra del codice oppure ci si sposta dalla linea del codice con i tasti freccia o col cursore del mouse. Se la finestrella delle informazioni rapide automatiche anziché essere utile dà fastidio, la si può disabilitare (e riattivare) col comando **Strumenti - Opzioni dell'Editor VB**. Una volta immesso il codice sorgente della procedura Mondo la si può eseguire in questo modo:

- Selezionare il comando **Strumenti - Macro** per aprire la finestra di dialogo delle macro
- Selezionate la procedura *Mondo* dall'elenco dei nomi delle macro
- Fare clic sul pulsante Esegui

Quando il VBA esegue la procedura visualizza la finestra di dialogo sotto riportata

Fig. 2

Si consideri l'uso della funzione *MsgBox* a puro titolo di esempio, tratteremo questa interessante funzione molto dettagliatamente nel proseguo del corso, lo scopo della procedura Mondo era solo per mostrare i passaggi da eseguire per scrivere una procedura manualmente

Utilizzare il controllo ListView

Il controllo **ListView** viene utilizzato per visualizzare informazioni di tipo gerarchico e consente di mostrare elenchi o liste di dati, oltre ad essere di grande impatto visivo. Questo tipo di controllo è diventato popolare con Windows Explorer in quanto è lo stesso tipo di lista usata dall'interfaccia di Explorer (Gestione risorse o Esplora risorse) per visualizzare Files e cartelle, inoltre le sue proprietà permettono di personalizzarne la visualizzazione in quattro stili diversi che sono:

- *LargeIcon* (Icone grandi): Ogni elemento è visualizzato con un'icona e del testo in basso
- *SmallIcon* (Icone piccole): Ogni elemento è visualizzato con una piccola icona e del testo alla sua destra.
- *Details* (Dettagli): Gli elementi sono disposti uno per riga. Ogni riga è suddivisa in più colonne, la prima contiene l'elemento stesso, le altre visualizzano i suoi attributi, inoltre ogni colonna ha un'intestazione.
- *List* (Elenco): Ogni elemento è visualizzato con una piccola icona e del testo alla sua destra e gli elementi sono organizzati in colonne senza intestazione.

Per rendersi conto di come viene visualizzata ciascuna modalità basta aprire Windows Explorer e selezionare i comandi corrispondenti nel menu Visualizza e per darvi un'idea della flessibilità di questo controllo dovete sapere che il desktop di Windows non è nient'altro che un grande controllo ListView in modalità Icon, con sfondo trasparente.
Il controllo ListView è un componente aggiuntivo **.ocx** che può essere aggiunto a Visual Basic attraverso Windows Common Control 6.0 e per averlo disponibile in VB per Excel si deve seguire il percorso dal menu **Strumenti – Controlli aggiuntivi** e nella finestra che appare scorrere la lista e selezionare la voce **Microsoft Listview Control, versione 6.0**.

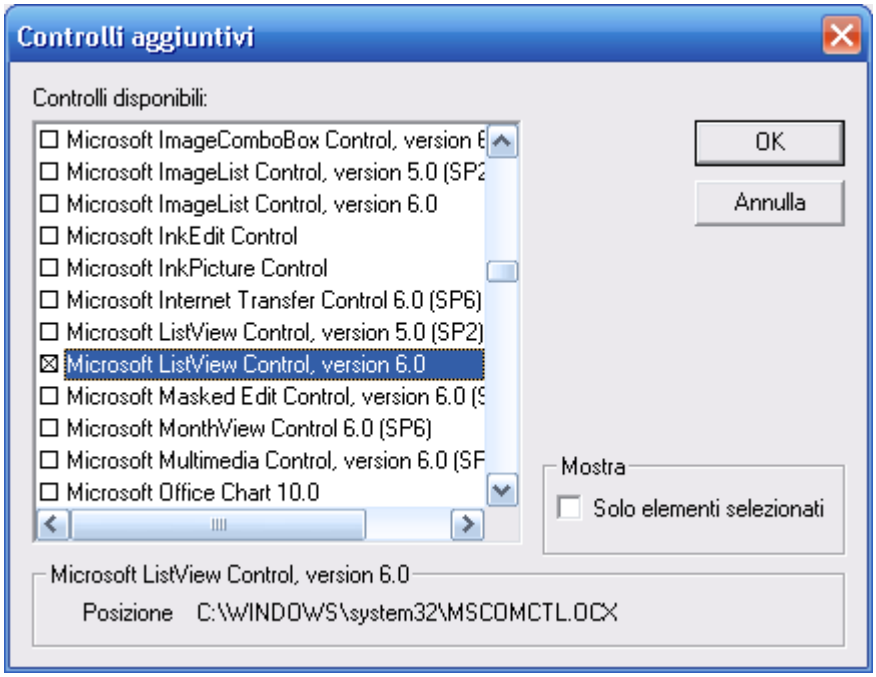


Fig. 1

Una volta confermata la scelta premendo sul pulsante Ok nella casella degli strumenti apparirà l'icona del controllo ListView

Fig. 2

Il controllo ListView dispone di due collection distinte:

- *ListItems* che comprende gli elementi testuali e grafici che rappresentano le voci da visualizzare e in modalità Report, è possibile specificare, per ogni voce, tutte le rispettive sottovoci mediante l'array ListSubItems.
- *ColumnHeaders* che include oggetti che influenzano l'aspetto delle singole intestazioni delle colonne visibili in modalità Report.

Esaminiamo ora la proprietà ListView che può essere: Icon View, Small Icon View, List View e Report View e per impostare una modalità di visualizzazione si deve assegnare alla proprietà View uno dei seguenti valori costanti:

0-lvwIcon = Icon View
1-lvwSmallIcon = Small Icon View
2-lvwList = List View
3-lvwReport = Report View

Per aggiungere un elemento al controllo ListView si deve usare il **metodo Add()** della collezione *ListItems*, tenendo presente che ogni riga di un oggetto ListView può essere definito in due parti

ListView1.ListItems(x): Specifica la riga per la prima colonna della riga
ListView1.ListItems(x).ListSubItems(y): Consente di specificare le colonne adiacenti, per esempio: *ListView1.ListItems(5).ListSubItems(1)* indica la seconda colonna della quinta riga della ListView

Per cui la sintassi per aggiungere **una riga** è la seguente: *ListView1.ListItems.Add [Index], [Key], [Text], [Icon], [SmallIcon]* , in cui le voci rappresentano:

[Index]: Valore Opzionale
È un numero che indica la posizione della voce all'interno della collezione. Se viene omesso, la voce viene inserita in coda alla collezione. Il suo valore varia in funzione di successivi inserimenti e cancellazioni
[Key]: Valore Opzionale
E' una stringa che rappresenta la chiave identificativa univoca di ogni voce nella collezione. Utile per la ricerca di una voce.
[Text]: Valore Opzionale
Rappresenta il testo che viene mostrato nel controllo, eventualmente associata ad una icona.
[Icon]: Valore Opzionale
Specifica l'immagine da visualizzare quando il ListView è la modalità lvwIcon
[SmallIcon]: Valore Opzionale
Specifica l'immagine da visualizzare quando ListView è lvwSmallIcon, lvwList o in modalità lvwReport

Mentre invece la sintassi per aggiungere un elemento alla colonna di destra della riga specificata è la seguente: *ListView1.ListItems(1).ListSubItems.Add [Index], [Key], [Text], [ReportIcon], [TooltipText]* , dove:

1:
Specifica il numero della riga nel controllo ListView.
[Index]: Valore Opzionale
Specifica il numero di colonna per l'aggiunta di un dato. Il valore 1 corrisponde alla seconda colonna di un oggetto ListView
[Key]: Valore Opzionale
E' una stringa che rappresenta la chiave identificativa univoca di ogni voce nella collezione
[Text]: Valore Opzionale
Specifica il testo che verrà visualizzato nel ListView
[ReportIcon]: Valore Opzionale
Visualizza un'icona o un'immagine in base all'elemento specificato
[TooltipText]: Valore Opzionale
Aggiunge un tooltip nell'elemento specificato

Mentre invece per aggiungere delle **colonne**, è necessario prima definire i testi, le dimensioni e le intestazioni usando la seguente sintassi:*ListView1.ColumnHeaders.Add [Index], [Key], [Text], [Width], [Alignment], [Icon]*, dove le voci rappresentano:

[Index]: Valore Opzionale
[Key] Valore Opzionale .
E' una stringa che rappresenta la chiave identificativa univoca di ogni voce nella collezione
[Text]: Valore Opzionale .
Specifica il testo che verrà visualizzato nel ListView.
[Width]: Valore Opzionale .
Specifica la larghezza della colonna. Il valore predefinito è 72 punti
[Alignment]: Valore Opzionale .
Specifica l'allineamento della colonna. Le costanti disponibili:. LvwColumnLeft (Default) lvwColumnCenter, lvwColumnRight
[Icon]: Valore Opzionale .
Specifica l'immagine da visualizzare nell'intestazione.

L'esempio che segue mostra il principio di riempimento di un oggetto ListView.

```
Private Sub UserForm_Initialize()
    With ListView1
        'Imposta il numero di colonne e intestazioni
        With .ColumnHeaders
            'Rimuove le vecchie intestazioni
            .Clear
     'aggiunge 3 colonne specificando il nome dell'intestazione e la larghezza della colonna
            .Add , , "Nome", 80
            .Add , , "Città", 50
            .Add , , "Età", 50
        End With
       'Riempie la prima colonna con 3 righe
        With .ListItems
           .Add , , "Gino"
           .Add , , "Mario"
           .Add , , "Elisa"
         End With
      'Riempie le colonne 2 e 3 della prima riga
        .ListItems(1).ListSubItems.Add , , "Città1"
        .ListItems(1).ListSubItems.Add , , 30
       'Riempie le colonne 2 e 3 della seconda riga
        .ListItems(2).ListSubItems.Add , , "Città2"
        .ListItems(2).ListSubItems.Add , , 27
        'Riempie le colonne 2 e 3 della terza riga
        .ListItems(3).ListSubItems.Add , , "Città3"
        .ListItems(3).ListSubItems.Add , , 41
    End With
  'Specifica le modalità di visualizzazione in "Dettagli"
    ListView1.View = lvwReport
End Sub
```

E si ottiene una Form come la seguente

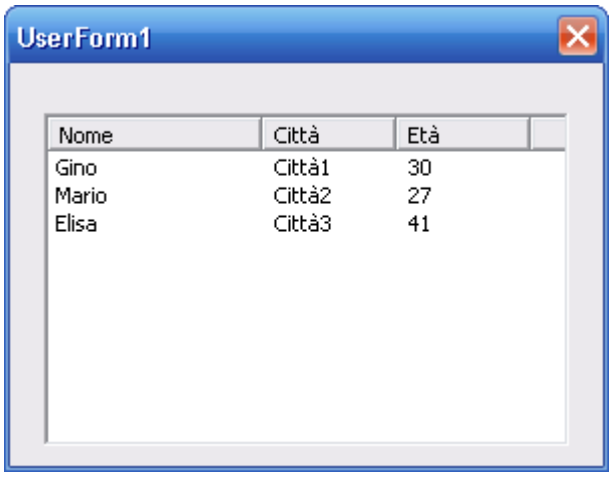


Fig. 3

Questa macro è un esempio semplificato ed è ovviamente possibile creare un loop per ottimizzare il riempimento. Una volta visualizzati nel controllo, i dati possono essere letti e modificati, la procedura sotto riportata scorre tutto il ListView e trasferisce le informazioni in un foglio di calcolo.

```
Private Sub CommandButton1_Click()
    Dim i As Integer, j As Integer
   'ciclo su tutte le righe
    For i = 1 To ListView1.ListItems.Count
        Cells(i, 1) = ListView1.ListItems(i).Text
       'Loop sulle colonne
        For j = 1 To ListView1.ColumnHeaders.Count - 1
            Cells(i, j + 1) = ListView1.ListItems(i).ListSubItems(j).Text
        Next j
    Next i
End Sub
```

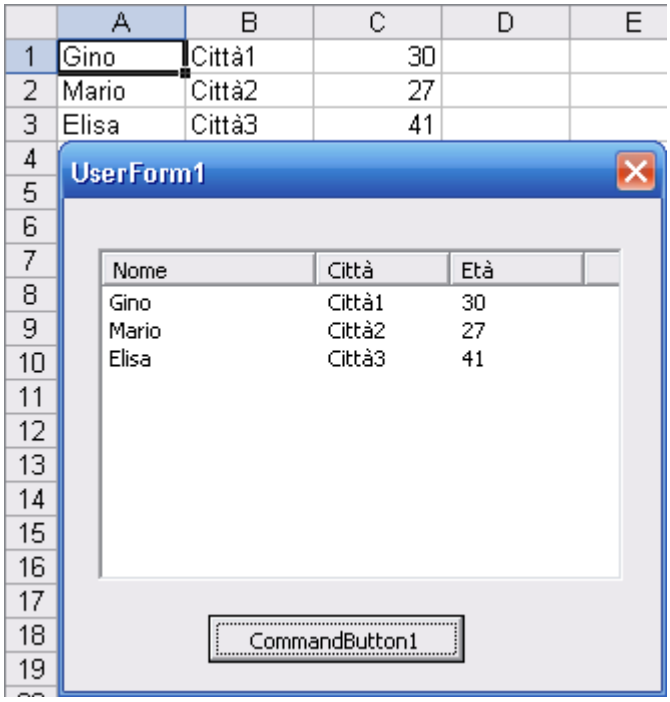


Fig. 4

Le informazioni contenute in una ListView possono essere facilmente modificate, ad esempio, se vogliamo modificare il testo nella terza colonna della prima riga possiamo usare un codice come il seguente

```
ListView1.listItems(1).listSubItems(2).Text = "Prova"
```

Un altro esempio per cambiare il testo nella terza riga della prima colonna

```
ListView1.ListItems(3).Text = "prova Modifica"
```

I dati nella prima colonna possono essere modificati anche manualmente nel controllo ListView, ma è possibile impedire questa azione specificando il valore 1 (lvwManual) nella **proprietà LabelEdit**. Oppure tramite questo codice

```
ListView1.labeledit = 1
```

E' inoltre possibile assegnare elementi chiave unici in una ListView in modo che i dati possono essere recuperati tramite questa chiave di identificazione. Per esempio può essere recuperato il contenuto della voce a cui viene assegnato il tasto "A1". Nota: la procedura restituisce un errore se la chiave non esiste nel ListView.

```
MsgBox ListView1.ListItems("A1").Text
```

E per recuperare una specifica voce sotto "A2" nell'elemento "A1"

```
MsgBox ListView1.ListItems("A1").ListSubItems("A2").Text
```

È inoltre possibile ottenere la chiave di una riga, questa procedura restituisce una stringa vuota se non è stato assegnato nessun tasto.

```
MsgBox ListView1.ListItems(2).Key
```

Questa macro consente di assegnare una chiave per la ListItem della seconda riga e se la chiave esiste già per questo elemento, verrà sovrascritto. Se si tenta di assegnare una chiave già assegnata ad un altro elemento, la procedura restituisce un messaggio di errore, e questo è logico in quanto la chiave deve essere univoca.

```
Private Sub CommandButton2_Click()
   'Legge la chiave originale
    MsgBox ListView1.ListItems(2).Key
   'Assegnare una nuova chiave al ListItem della seconda riga
    ListView1.ListItems(2).Key = "Nuova Key"
    'verifica  delle  nuove  chiavi
    MsgBox ListView1.ListItems(2).Key
End Sub
```

Inoltre è possibile eliminare delle righe specifiche nella ListView.

```
  'Rimuovere la terza riga
ListView1.ListItems.Remove 3
'altro esempio per eliminare una riga in base alla sua Key
ListView1.ListItems.Remove "A1"
'Eliminare la riga attiva
ListView1. ListItems. Remove (ListView1. SelectedItem. Index)
```

Mentre invece per eliminare tutti i dati in un controllo ListView si usa questo codice:

```
  ListView1.ListItems.Clear
```

E possibile modificare la formattazione del testo del controllo per personalizzare la visualizzazione delle informazioni, nel codice sotto riportato si modifica il colore del testo nel 2 ° elemento della prima riga.

```
ListView1.listitems(1).ListSubItems(2).ForeColor = RGB(100, 0, 100)
```

Oppure si può applicare un formato a una "cella" del listview

```
ListView1.ListItems(2).ListSubItems.Add , , Format(1234567.89, "##,##0.00")
```

Inoltre tramite la proprietà *FullRowSelect* si può evidenziare l'intera riga in una selezione.

```
ListView1.FullRowSelect = True
```

La proprietà *Griglia* permette di visualizzare una griglia nel ListView, questa proprietà è molto utile per migliorare la leggibilità dei dati

```
ListView1.Gridlines = True
```

Un'altra opzione del controllo consente di visualizzare le caselle di controllo nella colonna di sinistra.

```
Me.ListView1.CheckBoxes = True
```

È quindi possibile specificare lo stato di default del CheckBox, se non si specifica questo parametro, il testo non sarà visibile subito, ma è necessario fare clic sul bordo sinistro della riga per far apparire il CheckBox.

```
Dim i As Integer
For i = 1 To ListView1.ListItems.Count
   ListView1.ListItems(i).Checked = False
Next i
```

Possiamo usare l'evento *ItemCheck* per identificare quando una casella è selezionata oppure deselezionata e modificare il colore del testo in blu e grassetto.

```
Private Sub ListView1_ItemCheck(ByVal Item As MSComctlLib.ListItem)
   Dim j As Integer
   If Item.Checked = True Then
         'Cambia Colore
         Item.ForeColor = RGB(0, 0, 255)
         'Imposta il grassetto
         Item.Bold = True

         For j = 1 To Item.ListSubItems.Count
            Item.ListSubItems(j).ForeColor = RGB(0, 0, 255)
            Item.ListSubItems(j).Bold = True
         Next j
      Else
          'Cambia Colore
         Item.ForeColor = RGB(1, 0, 0)
         Item.Bold = False

         For j = 1 To Item.ListSubItems.Count
            Item.ListSubItems(j).ForeColor = RGB(1, 0, 0)
            Item.ListSubItems(j).Bold = False
         Next j
   End If
End Sub
```

Inoltre è possibile scegliere di nascondere le intestazioni delle colonne utilizzando la proprietà *HideColumnHeaders*.

```
ListView1.HideColumnHeaders = True
```

La seguente macro specifica che i dati devono essere centrati nella colonna

```
ListView1.ColumnHeaders.Add , , "Città", 50, lvwColumnCenter
```

La struttura permette tramite la proprietà *AllowColumnReorder* di spostare la posizione delle colonne tramite un drag & drop.

```
ListView1.AllowColumnReorder = True
```